UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

TRABAJO FIN DE GRADO

Desarrollo de la lógica de una aplicación en Android y de la integración de la misma con un entorno físico

Bruno Follon
Tutor: Simone Santini

Junio 2014

RESUMEN

En el presente documento se detallan las fases de diseño y desarrollo del proyecto *"Desarrollo de la lógica de una aplicación y de la integración de la misma con el entorno físico"*. A lo largo del documento se ha intentado dar toda la información necesaria al lector para asegurar el entendimiento del mismo.

El concepto tras la aplicación es el de un software diseñado para funcionar en conjunción con un juego de mesa, sirviendo de apoyo al mismo. Esta idea surge como noción para conciliar la tecnología y la interacción social de las personas. Si bien hoy en día el teléfono móvil parece alejarnos a cada vez más del mundo que nos rodea, este proyecto intenta hacer uso de esta tecnología para realizar todo lo contrario: juntar a varios amigos alrededor de una mesa para disfrutar de una partida de un juego de mesa.

Este documento trata las distintas ideas recogidas en el proceso de creación y diseño de la aplicación, así como los múltiples problemas encontrados en el camino.

Conciliar un concepto tan tradicional como un juego de mesa con uno tan actual como una aplicación no es tarea fácil, es muy tentador dejarse llevar por alguno de los dos extremos y generar un producto que finalmente no se corresponde con la idea inicial. A lo largo del documento veremos que surgen múltiples problemas de integración y equilibrio entre ambas partes del conjunto.

Ha sido preciso extensa tarea de diseño para solventar esta unión de conceptos, llegando al punto de tener que forzarse a limitar los ámbitos tocados por la aplicación y buscando soluciones que, a primera vista, son lejos de ser óptimas.

En este documento se aporta, además, una explicación de las tecnologías usadas para el desarrollo del mismo. A su vez podrán encontrarse una serie de anexos con contenido no directamente relevante para el proyecto pero no por ello menos relacionado con el mismo ni menos interesante.

Se espera que el estilo seguido en el documento sea de agrado al lector y convierta la tarea de lectura en algo más ameno.

ABSTRACT

In this paper the design and development phases of the project "*Desarrollo de la lógica de una aplicación y de la integración de la misma con el entorno físico*" are detailed. Throughout the paper, an effort to give all the information necessary to the user's comprehension has been made.

The idea behind this application is that of software designed in order to work in conjunction with a board game, helping the latter. This concept appears as a way to conciliate technology with social interaction. Nowadays mobile phones are often seen as something that makes us less aware of our surroundings; this project tries to make use of this technology in a completely different fashion: we want to gather a bunch of friends together in order to have a great time playing a board game.

This paper goes through the various ideas contained in the process of creation and design of the application, and it also discusses the many problem encountered along the way.

Reconciling a traditional concept such as a board game with one as current as an application is no easy task; it is very tempting to go with one of the two ends and finally end with a product that is far away from the initial idea. Throughout the paper we will see many problems arising from integration and balance between the two part of whole.

An extensive design task has been needed in order to solve this union of concepts, to the point of having to limit the areas the application reaches and looking for solution that, at first glance, are far from optimal.

This document also provides an explanation of the technologies used for development. At the end of this document, a series of annexes can be found; such annexes may not be directly relevant to the project but they surely are related to it and are no less interesting in its contents.

It is expected that the style followed in the document will be pleasing to the reader and may make the reading task more enjoyable.

PALABRAS CLAVE

En este apartado se incluyen una serie de palabras clave con el fin de indexar el proyecto.

- Aplicación
- Android
- Java
- QR
- Móvil
- Smartphone
- Juego de mesa
- Ámbito físico

KEYWORDS

A set of keywords are included in this section for the sake of indexing the project.

- Application
- Android
- Java
- QR
- Mobile
- Smartphone
- Board game
- Physical environment

TABLA DE CONTENIDO

ÍNDICE DE ILUSTRACIONES

INDICE DE TABLAS

INTRODUCCIÓN

MOTIVACIÓN

Escoger un tema sobre el cual realizar un trabajo de fin de carrera no es tarea fácil, es preciso que dicho tema sea lo suficiente interesante y ameno como para la llevar a cabo la tarea en sí y a esto hay que sumarle que tenga un cierto valor académico y de investigación.

Por suerte, estos ingredientes no tienen por qué estar reñidos, es más, a menudo el conocimiento obtenido a través de la investigación resulta lo suficientemente gratificante como para cumplir con lo propuesto.

El tema escogido en este caso es el desarrollo de una aplicación de Android. “Otra más” podría pensarse, y no se erraría en exceso, sin embargo hay ciertos detalles que nos permiten distinguir a este árbol en medio del bosque. Estos detalles fundamentalmente se deben al enfoque de la aplicación.

El proyecto consiste en desarrollar una aplicación en conjunto con un juego de mesa, siendo la aplicación una forma de apoyo a éste. Esta idea, que en principio puede parecer descabellada, plantea diversos e interesantes problemas en cuanto a la integración de la aplicación con el entorno (forzosamente físico) del juego de mesa. A su vez, también se plantean numerosos problemas de equilibrio y diseño entre ambas partes del conjunto, que se detallarán más adelante. Como pequeño adelanto, destacaremos que diseñar un hibrido entre juego y aplicación implica tener un cuidado extremo en el reparto del protagonismo a cada parte de manera que el producto final deje una impresión de mezcla adecuada en los jugadores.

Otro factor influyente es el hecho de que es una idea relativamente poco explotada, lo que da lugar a posibles oportunidades en el mercado. Aunque cuando se piense en el mercado de los juegos de mesa no nos venga “Sillicon Valley” a la cabeza, no deja de ser un mercado activo en el que sin ir más lejos que 2014 podemos ver numerosos y pintorescos proyectos saliendo a la luz [1].

OBJETIVOS

Con este proyecto se pretende diseñar e implementar una aplicación de apoyo a un juego de mesa en dispositivos Android.

La razón por la cual se escoge la plataforma Android en lugar de iOS o WindowsPhone es meramente práctica en el sentido en que esta plataforma es la más extendida entre los Smartphones alcanzando un 80% de la cuota de mercado en 2014 [2].

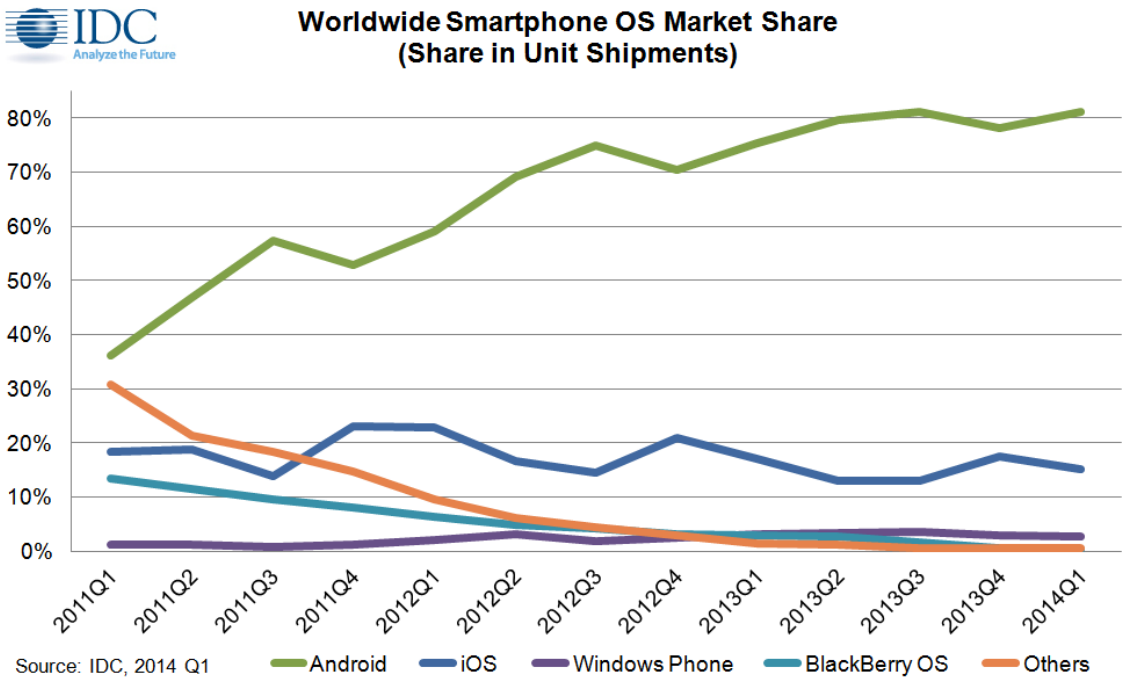

ILUSTRACIÓN 1: MOBILE OS MARKET SHARE [2]

Es importante destacar que se quiere lograr una integración de la aplicación con el entorno físico del juego de mesa con las limitaciones y problemas que eso conlleva; Se estudiarán dichas limitaciones y problemas a fondo más adelante en la sección [Equilibrio entre juego de mesa y aplicación]

Se pretende dar soporte al mayor número de versiones de Android posible probando la aplicación en la API de Android de menor nivel a la que se pretende dar soporte; Actualmente dicha API es la 14. Para conseguir una aplicación retro-compatible, es preciso prestar atención a las funcionalidades de las que se hace uso.

ESTRUCTURA DEL DOCUMENTO

Este documento se divide en varios apartados, siendo el primero la introducción.

A continuación le sigue el Estado del arte, en el que se investigan las tecnologías utilizadas.

Seguido a esto se encuentra el análisis de requisitos, en el que mediante reuniones con el cliente se analizan los requisitos funcionales y no funcionales de la aplicación y se describen una serie de posibles casos de uso para el prototipo sobre el que trata este proyecto.

En la siguiente sección se detalla el diseño de la solución aportada para el proyecto, incluyendo esto los patrones seguidos para implementar la solución, un diagrama de clases y ciertos aspectos de diseño de la aplicación.

A continuación está el apartado de implementación, en el que se detallan los algoritmos creados para la realización de este prototipo, así como aspectos más técnicos de diseño.

En el apartado siguiente, conclusiones, el lector podrá encontrar una conclusión sobre el proyecto.

La bibliografía se sitúa después de las conclusiones.

Finalmente se encuentran los anexos en los que el lector podrá encontrar asuntos que no tenían legítima cabida en el documento del proyecto pero que sin embargo se consideran interesantes para una mayor comprensión del mismo.

JAVA [1]

En esta sección nos adentraremos levemente en el lenguaje de programación que hemos usado para desarrollar nuestra aplicación Android, es decir, Java.

Java es un lenguaje de programación concurrente, basado en clases, orientado a objetos y específicamente diseñado para ser escasamente dependiente de la implementación. Se pretende que una aplicación desarrollada en java se codifique una vez y se pueda ejecutar prácticamente en todas partes.

Un programa escrito en Java es compilado en lo que se denomina "Bytecode" de Java. Esto implica que un programa en Java puede ejecutarse en cualquier máquina virtual de java (JVM) independientemente de la arquitectura de la máquina en la que se esté ejecutando.

JAVA BYTECODE [2]

Un programa escrito en Java es compilado a Bytecode, tal y como se ha explicado antes. Vamos a ahondar ligeramente en este concepto con el fin de aclararlo al lector.

La compilación realiza una traducción de las instrucciones legibles por humanos a código máquina. En el caso de Java, el bytecode es compuesto por uno o dos bytes que representan la instrucción de máquina a ejecutar seguido por cero o más bytes de parámetros.

Con un byte por instrucción (en su mayoría) tenemos 256 posibles instrucciones. Actualmente no son usadas las 256 posibilidades, de hecho existen tres valores que están marcados para no ser implementados nunca.

No vamos a poner una lista con las posibles instrucciones existentes en Java Bytecode, el lector que desee una lista de ese tipo la puede encontrar en [3].

PROGRAMACIÓN CONCURRENTE [3]

Se conoce como programación concurrente a aquella programación en la que varias instrucciones son ejecutadas en un mismo periodo de tiempo.

[1] [25] [20]
[2] [21]
[3] [18]

Un sistema es concurrente cuando es capaz de ejecutar varias instrucciones sin tener que "esperar" a que otras se completen.

Es importe diferenciar entre computación concurrente y computación paralela, ya que es sencillo confundirlas.

La computación paralela es aquella en que varios procesos se ejecutan literalmente al mismo tiempo. Esto sólo es posible en procesadores con más de un núcleo ya que sólo una instrucción puede darse por cada ciclo de reloj del núcleo.

La computación concurrente, sin embargo, se define como aquella en la cual varios procesos son ejecutados con ciclos de vida superpuestos. Esto significa que tal computación si es posible en procesadores de un único núcleo ya que un proceso puede ser detenido a la mitad y puede empezar otro. De esta forma un proceso podrá estar "pausado" mientras otros toman el turno para ejecutarse. Esto se suele conseguir asignando franjas de tiempo a cada proceso.

PROGRAMACIÓN ORIENTADA A OBJETOS [4]

La programación orientada a objetos es un paradigma de programación que representa el concepto de objeto.

En ingeniería informática, un objeto es una posición de memoria con un valor y que puede ser referenciada mediante un identificador. En la programación orientada a objetos un objeto tiene atributos (campos de datos) que describen el objeto y una serie de funciones asociadas con dicho objeto conocidas como métodos.

La programación orientada a objetos está diseñada para conseguir un software modular reutilizable. El término modular hace referencia a algo que funciona "por sí solo" y que puede ser modificado o ampliado sin afectar a aquello que haga uso de dicho módulo. Este paradigma de programación es una evolución de previas técnicas como la programación estructurada y la abstracción de tipos.

Los objetivos de la programación orientada a objetos son:

- Mayor comprensión del código
- Facilidad de mantenimiento
- Facilidad de evolución

Numerosas técnicas son usadas para lograr dichos objetivos; Destacaremos el concepto de **encapsulación** y el de **ocultación de la información**.

[4] [19]

La encapsulación consiste en restringir el acceso a componentes del objeto de manera que estos sólo sean modificados como lo desee el desarrollador. Pongamos un ejemplo, imaginemos que tenemos un objeto con varios atributos.

Ahora de estos atributos supongamos que tenemos un atributo que depende del valor de los demás pero que en ningún caso podrá ser modificado directamente. Mediante la encapsulación podemos conseguir precisamente esto; Si el desarrollador lo desea, restringirá el acceso a dicho atributo y no creará ningún método que permita la modificación directa del mismo.

Pasemos a continuación al concepto de ocultación de información. La ocultación de información consiste en ocultar las decisiones de diseño de un programa tomadas cuando éstas son susceptibles de cambio. Esto se hace con el fin de proteger otras partes del programa cuando dicho diseño cambia. Con esto se consigue aislar la implementación del programa creando una interfaz estable.

Ambos conceptos son similares dado que persiguen una idea parecida: crear un software cuyos detalles internos están protegidos del exterior.

Con esto terminaremos nuestra breve exposición sobre los lenguajes de programación orientados a objetos.

PROGRAMACIÓN BASADA EN CLASES [5]

La programación basa en clases, usada por Java, es un estilo de programación orientada a objetos en el que la herencia se consigue mediante clases de objetos en vez de los objetos en sí.

Dichas clases definen la estructura y comportamiento de los objetos. De esta forma una clase es un poco como el esqueleto con la cual se hacen los objetos. Los objetos pasan a ser, pues, instancias de las clases.

[5] [17]

ANDROID[6]

¿QUÉ ES?

Android es un sistema operativo para dispositivos móviles basado en el kernel de Linux. Actualmente desarrollado por google, Android tiene una interfaz basada en la interacción directa, esto es, está especialmente diseñado para pantallas táctiles de móviles, tabletas, etc. Android también provee interfaces especiales para coche, televisión y relojes.

Android intenta hacer uso de gestos con una vaga correspondencia a acciones que podrían realizarse en el mundo real como deslizar el dedo, tocar y pellizcar para el manejo de objetos en pantalla.

Como ya se ha dicho previamente, Android tiene una cuota de mercado muy superior a la de sus competidores, vendiendo más dispositivos que los demás sistemas operativos juntos [4].

El código fuente de Android es publicado bajo licencia de código abierto por Google, aunque la mayoría de proveedores venden sus dispositivos con versiones privadas de Android [5].

La naturaleza abierta de Android lo ha convertido en la alternativa más escogida por desarrolladores para publicar aplicaciones y modificaciones del sistema operativo, creando una comunidad de entusiastas a su alrededor.

HISTORIA

Android, inc. fue fundada en Palo Alto, California en Octubre de 2003 por Andy Rubin, Rich Miner, Nick Sears y Chris White con el fin de desarrollar "Dispositivos móviles más inteligentes y conscientes de las preferencias y localización del usuario". La idea original de Android era desarrollar un sistema operativo para cámaras digitales; Sin embargo, tras comprobar la falta de un mercado suficiente, decidieron reorientar el proyecto a un sistema operativo para dispositivos móviles con el fin de competir contra Symbian y Windows Mobile.

Google compró Android inc. el 17 de agosto de 2005, manteniendo a los empleados más significativos.

El 5 de Noviembre de 2007, la "Open Handset Alliance", un consorcio de compañías entre las que se encontraba Google desveló su intención de desarrollar un estándar

[6] [27]

abierto para dispositivos móviles, presentando Android como su primer proyecto, construido sobre la versión 2.6.25 del kernel de Linux.

El primer dispositivo Android en salir al mercado fue lanzado el 22 de octubre de 2008.

En la actualidad y desde 2010 Google saca su propia gama de dispositivos Android bajo el nombre de Nexus. Estos dispositivos incluyen móviles y tabletas. Google utiliza dichos dispositivos como referencia para demostrar las capacidades de Android.

VERSIONES

Android ha recibido desde 2008 numerosas actualizaciones de seguridad, arreglo de errores y añadido de funcionalidad. Dichas versiones son nombradas en orden alfabético y siempre referencian un postre azucarado como Donut, KitKat o Cupcake.

La versión más reciente de Android es 4.4 KitKat, que hace uso de la API 19 de Android.

Sin embargo no es la versión más usada actualmente [6], como se puede comprobar en la siguiente figura:

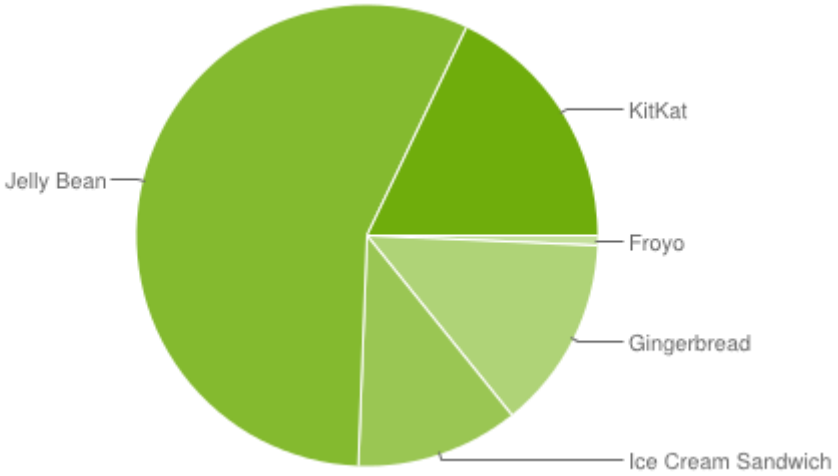

ILUSTRACIÓN 2: USO DE VERSIONES DE ANDROID [6]

Esto se debe a que los usuarios no necesariamente desean actualizar sus dispositivos y a que a menudo dicha actualización depende de que el proveedor del dispositivo lance su propia versión personalizada de Android.

¿CÓMO FUNCIONA?

En esta sección haremos una breve exposición sobre algunos de los mecanismos de Android que se consideran relevantes para el entendimiento de este proyecto.

LAYOUTS

El concepto de layout es lo primero que vamos a ver.

En Android, un layout es lo que se muestra al usuario por pantalla, un concepto parecido al de vista en el patrón de diseño modelo – vista – controlador.

Los layouts están codificados en formato XML y contienen la información sobre los distintos elementos que se van a mostrar por pantalla. Lo que interpreta los layouts es lo que llamamos un “inflater”.

Los ficheros de layout están guardados en la carpeta layout bajo la carpeta res, que almacena los recursos de la aplicación.

ACTIVITIES E INTENTS

En Android una activity o actividad es un código java que tiene una interfaz de usuario asociada. Podríamos decir, a grandes rasgos que cada pantalla de la aplicación es una actividad. Aunque esto es falso ya que una misma actividad puede "inflar" distintos layouts a lo largo de su ciclo de vida.

Este es el ciclo de vida de una actividad Android:

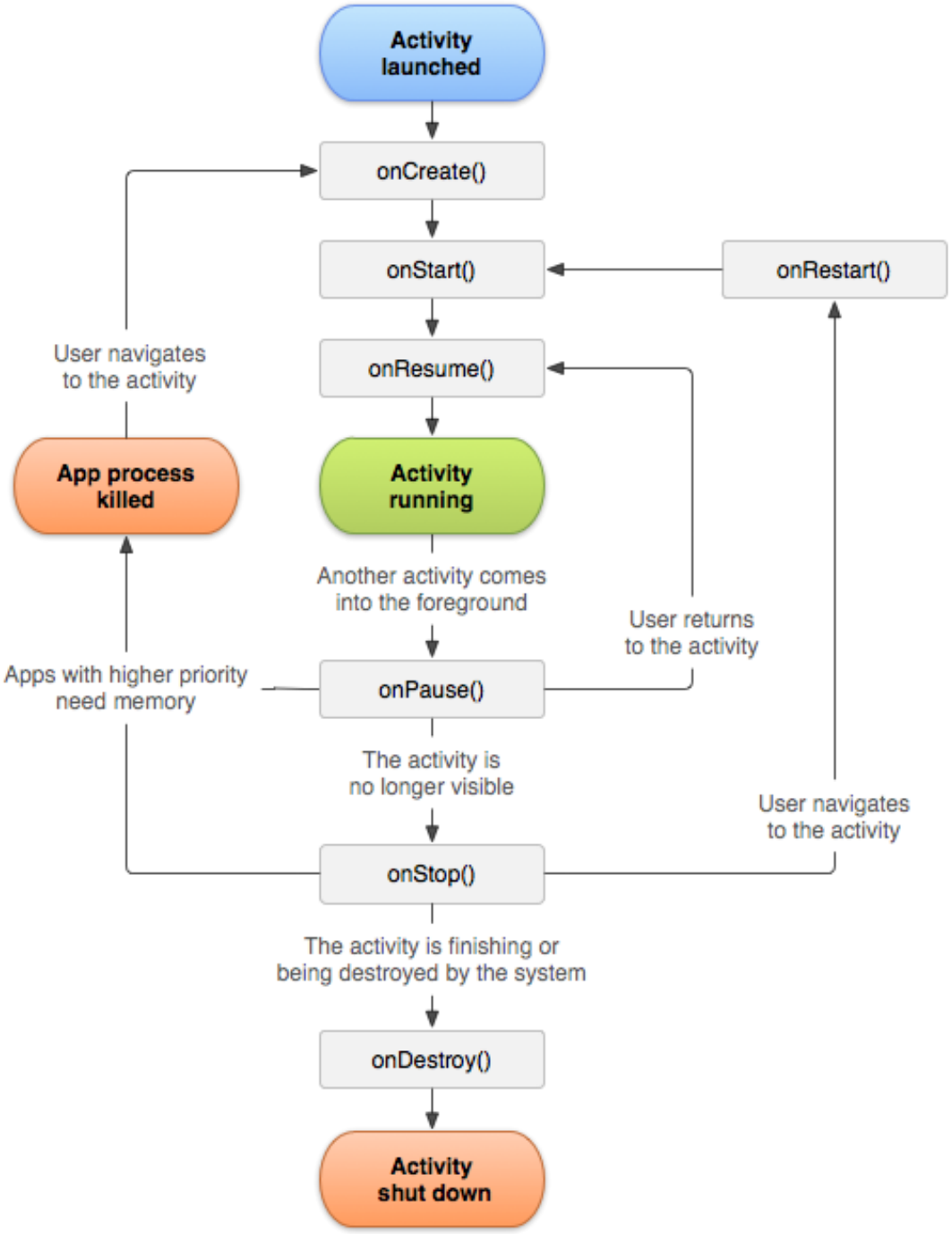

ILUSTRACIÓN 3: CICLO DE VIDA DE UNA ACTIVIDAD [7]

En la [Ilustración 3: Ciclo de vida de una actividad], podemos ver el ciclo de vida de cualquier actividad Android. Lo que vemos son los métodos que son llamados a lo largo de la vida de la actividad. Cuando se crea una actividad se llama a onCreate, onStart y onResume antes de que la actividad esté ejecutándose. En estos métodos es cuando se inflará el layout inicial de la actividad.

Cuando otra actividad ocupe el primer plano (literalmente), la actividad se pausará mediante el método onPause. Si la actividad deja de estar visible por completo se llamará a onStop. Si, mediante interacción del usuario, la aplicación vuelve a estar visible, se llamará a onRestart() siempre y cuando el proceso de la actividad no haya sido terminado por el procesador por falta de memoria. En caso de que se haya terminado el proceso, se llamará a onCreate.

Cuando la actividad termina se llama a onDestroy y la actividad termina su ciclo de vida.

A través de todos estos métodos que se pueden ver en la [Ilustración 3: Ciclo de vida de una actividad], un programador puede controlar que ocurre con cada actividad en cualquier momento de su ciclo de vida. El desarrollador podrá guardar información sensible en caso de que el sistema operativo mate el proceso o se pierda el enfoque, etc.

INTENTS[7]

En cuanto a los intents, se usan para lanzar nuevas actividades. Para lanzar una nueva actividad es preciso crear un intent y llamar al método startActivity que recibe como parámetro el intent que acabamos de crear.

Es importante que después de lanzar una nueva actividad mediante un intent se termine la actividad actual (siempre y cuando esto sea lo pertinente)

NAVEGACIÓN ENTRE ACTIVIDADES: PILA DE ACTIVIDADES

Android mantiene una pila de actividades según se vayan creando de manera que cuando finaliza el ciclo de vida de una aplicación se pueda volver a la actividad anterior que esté pausada o parada.

Si se llama al método **finish** en una actividad, la entrada correspondiente a esta actividad en la pila de actividades se eliminará.

También es posible controlar si se desea vaciar la pila de actividades en algún momento. Por ejemplo si un usuario vuelve al menú principal desde un submenú que esté ubicado a tres actividades de la del menú principal, es interesante vaciar la pila de

[7] [31] [40]

actividades de manera a que si el usuario pulsa el botón atrás se cierre la aplicación en vez de volver al submenú previamente mencionado.

Esto se hace pasando un valor booleano a la función startActivity. Concretamente este valor: Intent.FLAG_ACTIVITY_CLEAR_TOP [8].

FICHERO ANDROIDMANIFEST.XML[8]

Toda aplicación Android debe aportar un fichero llamado AndroidManifest de extensión XML. En este fichero se guarda información de configuración de la aplicación. Algunos de los contenidos de este fichero son:

- El nombre del paquete Java para la aplicación, que hace la vez de identificador único para la misma.
- Describe los componentes de la aplicación.
- Determina qué procesos acogerán los componentes de la aplicación
- Señala los permisos que necesita la aplicación para funcionar y los permisos que necesitan otras aplicaciones para interactuar con la misma.
- Indica el nivel mínimo de API necesario para hacer funcionar la aplicación.

En el apartado de los componentes es donde se indican las distintas actividades de las que se compone la aplicación. Todas las actividades deben ir declaradas en el fichero de manifiesto, en caso contrario no podrán ser iniciadas.

No vamos a ahondar más sobre el funcionamiento de Android ya que podríamos abarcar demasiadas páginas. Los conceptos aclarados en esta deliberadamente breve introducción sobre el funcionamiento de Android deberían bastar al lector para entender el resto del documento.

JUEGOS DE MESA QUE USEN APLICACIONES

Se han realizado búsquedas en internet de juegos de mesa de características similares a las de este proyecto en el sentido en que usen una aplicación como apoyo al "gameplay".

Se ha encontrado un concepto muy parecido en el sentido que acabamos de describir, sin embargo tiene diferencias fundamentales. Este concepto, financiado por crowdfunding [9] gracias a la plataforma kickstarter, tiene como idea simplificar los complicados juegos de mesa por medio de una aplicación. La aplicación que incorpora hace las veces libro de normas y también ayuda en ciertos aspectos como el movimiento de los personajes sobre el tablero y la resolución de conflictos [10].

[8] [30]

Si bien este concepto se asemeja bastante a este proyecto, existen diferencias muy marcadas. La primera y más sonada es que en este proyecto, cada jugador deberá usar la aplicación en su dispositivo móvil. Esto, que en principio puede parecer un impedimento dado que aumenta el número de dispositivos necesarios para jugar, convierte el teléfono de un jugador en ***su*** dispositivo. Es decir, cada jugador tendrá datos personales en su aplicación y gestionará información como le plazca.

Otra diferencia es que el concepto de la aplicación en este proyecto es radicalmente distinto. La idea de este proyecto es que la aplicación sea una parte más del juego, no una simple enciclopedia.

El objetivo de este proyecto es eliminar ciertos tiempos muertos característicos de los juegos de mesa; Sin embargo ahondaremos en estos conceptos más adelante en la sección [Equilibrio entre juego de mesa y aplicación]

LECTURA DE CÓDIGOS QR

Un aspecto muy importante en este proyecto es la lectura de códigos QR, ya veremos más adelante por qué es tan importante. En este apartado estudiaremos un poco los lectores de códigos QR.

QR Code es una abreviatura de "Quick Response Code", una marca registrada para designar códigos de barra de dos dimensiones (o en matriz). Inventados por Denso Waves en 1994, estaban originalmente diseñados para mantener un control sobre vehículos mientras se construían en fábrica.

Denso Waves aún mantiene la patente de los códigos, sin embargo ha liberado los códigos bajo licencia pública y hasta ha publicado las especificaciones online [11].

Hoy en día los códigos QR pueden ser leídos por los smartphones que se encargan de descodificar la información contenida en un código QR.

ESTRUCTURA Y FUNCIONAMIENTO

Si bien en sus inicios un código QR podía contener 4 caracteres de datos en un código de 21x21 píxeles, hoy en día los códigos pueden contener hasta 1817 caracteres de información en 177×177 píxeles.

Los códigos QR funcionan de manera parecida a los códigos de barra tradicionales en el sentido en que la luz reflejada en el código puede ser interpretada.

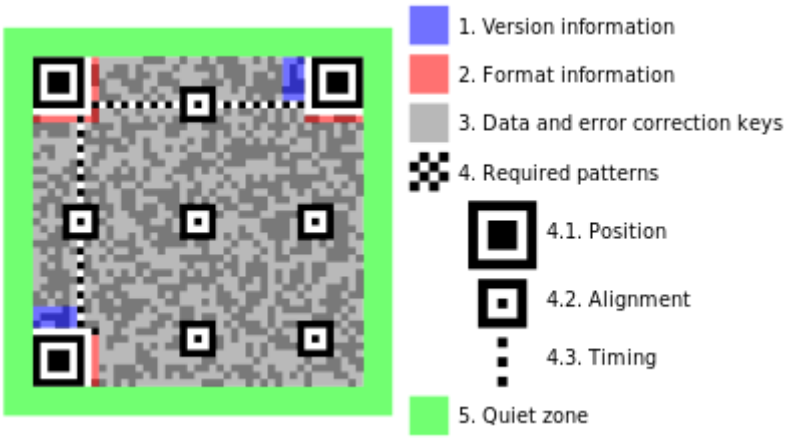

ILUSTRACIÓN 4: ESTRUCTURA DE UN CÓDIGO QR [12]

En la siguiente imagen podemos ver la estructura que sigue todo código QR.

Los cuadrados grandes sirven para el posicionamiento y los cuadrados más pequeños para la normalización del ángulo y tamaño de la lectura.

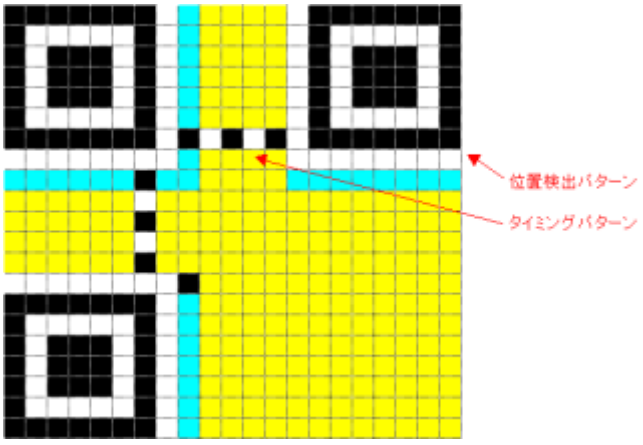

ILUSTRACIÓN 5: ÁREAS DE UN CÓDIGO QR [13]

En la [Ilustración 5: Áreas de un código QR] se puede ver más claramente las zonas del código. La zona amarilla representa dónde puede ir contenida información. Las bandas azules contienen información de formato y versión.

De entre las varias características de los códigos QR, una muy interesante es que incorporan un mecanismo de corrección de errores. Dependiendo del nivel de corrección de error, hasta un 30% del código puede ser restaurado en caso de estar dañado. Cuanto más alto es el nivel de corrección de error, menor es la cantidad de datos que se puede guardar.

La corrección de errores de los códigos QR se basa en el algoritmo de corrección de errores de Reed-Solomon [14].

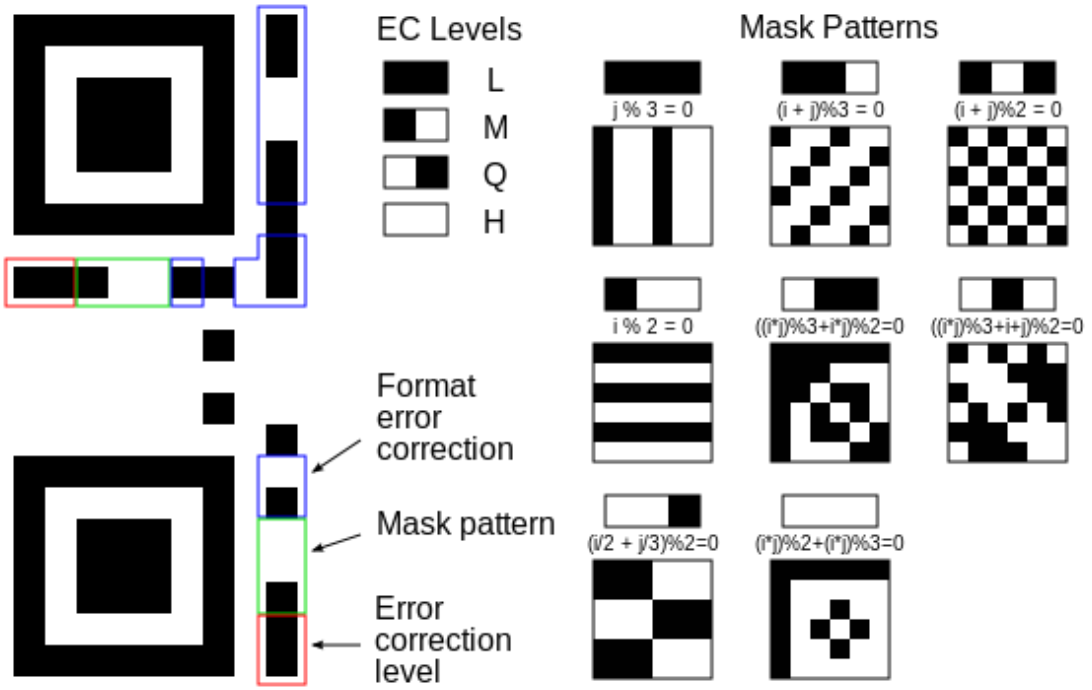

ILUSTRACIÓN 6: INFORMACIÓN DE FORMATO DE UN CÓDIGO QR [12]

En la [Ilustración 6: Información de formato de un código QR] podemos ver una aclaración de qué significa la información de formato previamente mencionada.

Como podemos ver, se almacena información sobre el patrón del código QR, el nivel de corrección de error y el nivel de corrección de errores del formato.

Terminaremos con esto nuestra breve exposición sobre los códigos QR. Pasaremos a continuación a explicar cómo integrar un lector QR a nuestra aplicación.

ANÁLISIS DE REQUISITOS

INTRODUCCIÓN

PROPÓSITO DEL SISTEMA

Se pretende diseñar e implementar para la plataforma Android™, una aplicación que sirva de apoyo a un juego de mesa.

La aplicación deberá integrarse con el juego de mesa de manera no intrusiva, permitiendo realizar acciones durante los periodos de inactividad mientras un jugador espera su turno. Es preciso que se cubran funciones de gestión de inventario, programas, combinación de objetos, gestión de los "puntos de vida" y "puntos de cordura" del usuario, gestión de usuarios localmente.

En una segunda fase, la aplicación deberá permitir funcionalidades de "pirateo" de terminales ajenos, envío de mensajes, gestión de conflictos entre usuarios, diseño de varias interfaces para la ocultación de información durante los conflictos y la fase de turno de un jugador, gestión de los turnos de la partida, gestión de usuarios online y gestión de partidas multijugador.

Es importante señalar que este proyecto se centra en la primera fase del proyecto, luego la segunda fase del proyecto no se implementará; Se indicará específicamente cuando una funcionalidad de la segunda fase sea implementada o diseñada.

ÁMBITO DEL SISTEMA

La aplicación deberá servir de apoyo durante una partida de juego de mesa, reduciendo los "tiempos muertos" que caracterizan a los mismos.

Con tiempos muertos nos referimos a los momentos en los que el jugador no está en su turno activo y en los que, tradicionalmente, sólo puede revisar su estado y analizar las jugadas del oponente.

Con esta aplicación se pretende abrir un abanico de posibilidades para el jugador en tiempo muerto tales como combinar objetos, gestionar su inventario, etc.

También se pretende dar uso a la aplicación durante los tiempos activos del jugador.

La aplicación deberá prestar apoyo al juego de mesa de la siguiente forma:

- Permitir el añadido de objetos y la gestión del inventario del jugador.
- Permitir la combinación de objetos dadas unas “recetas”.
- Gestionar los puntos de vida y de cordura del jugador.
- Gestionar las “habilidades” del jugador.

OBJETIVOS Y CRITERIOS DE ÉXITO DEL PROYECTO

Los objetivos principales del sistema, cuyo éxito es necesario para el proyecto son:

- Creación de partidas
- Añadido de objetos al inventario mediante escaneo de códigos QR
- Combinatoria de objetos del inventario dadas unas recetas
- Gestión de las habilidades del jugador

DEFINICIONES, ACRÓNIMOS Y ABREVIATURAS

- **Tiempo muerto:** Durante una partida del juego de mesa, se define como tiempo muerto a los tiempos en los que un jugador no está participando activamente en la partida, es decir, no está en su turno.
- **Tiempo activo:** Durante una partida del juego de mesa, se define como tiempo activo al tiempo durante el cual un jugador participa activamente en la partida, es decir, está en su turno.
- **Java:** Lenguaje de programación orientado a objetos. Será el lenguaje que se utilizará en la implementación de la aplicación.
- **App:** En este documento nos referiremos en ocasiones a conceptos como “Menú principal de la app”. Estos conceptos hacen referencia a las interfaces existentes por encima del juego. Esto es, existe una diferenciación entre las funcionalidades propias de la aplicación en sí (Como ajuste de volumen, vibración, perfil del usuario, etc.) que no se relacionan con la parte jugable de la aplicación.
- **Juego:** Con “juego” nos referiremos a las interfaces y mecanismos subyacentes a la instancia del juego. Es decir, cuando un jugador inicia una partida, entrará en el menú del juego y podrá realizar operaciones directamente relacionadas con el juego de mesa.

REQUISITOS FUNCIONALES

En esta sección analizaremos los requisitos funcionales de la aplicación.

ORGANIZACIÓN POR CARACTERÍSTICAS

1. **Registro de usuarios**
    a. **Introducción / Propósito de esta característica**

        Esta característica se basa en solicitar unas credenciales de inicio de sesión a los usuarios de la aplicación. En caso de no disponer de tales credenciales, la aplicación solicitará al usuario que se registre en la aplicación. Dado que en la primera fase del proyecto no se contempla funcionalidad online, este registro se hará localmente en el dispositivo. El propósito de este requisito es la escalabilidad de la aplicación de manera a no generar demasiados problemas cuando se pase a la segunda fase del proyecto.

    b. **Secuencia de estímulos / respuestas**

        El usuario inicia la aplicación. Después de cargar, la aplicación solicitará al usuario las credenciales de inicio de sesión. En caso de no disponer de credenciales el usuario podrá registrarse desde la propia aplicación.

        Una vez iniciada la sesión el usuario entrará en el menú principal de la app.

    c. **Requisitos funcionales asociados**
        i. **Requisito funcional 1: “Autenticación”**

            El usuario deberá autenticarse antes de poder usar la aplicación

        ii. **Requisito funcional 2: “Registro”**

            El usuario podrá registrarse desde la propia aplicación. Esta posibilidad se ofrecerá en la pantalla de autenticación.

2. **Creación de partida**
    a. **Introducción / Propósito de esta característica**

        Desde el menú principal de la app, el usuario podrá elegir iniciar una nueva partida. Esto le llevará al menú principal del juego.

b. Secuencia de estímulos / respuestas

El usuario autenticado en la aplicación escoge la opción de nueva partida e introduce un nombre y una contraseña para la partida. Esto le lleva al menú principal del juego.

3. Gestión del inventario

a. Introducción / Propósito de esta característica

Esta característica ilustra la capacidad del usuario de gestionar su inventario incluyendo esto añadido y borrado de objetos y combinatoria de objetos.

b. Secuencia de estímulos / respuestas

El usuario añade un objeto al inventario mediante escaneo de código QR o introducción de ID de objeto.

El usuario podrá borrar objetos del inventario en caso de que lo considere necesario.

El usuario también podrá intentar combinar diferentes objetos para crear un nuevo objeto directamente desde el inventario.

c. Requisitos funcionales asociados

i. Requisito funcional 1: "Añadir objeto mediante escaneo de código QR"

El usuario podrá en escanear códigos QR seleccionando dicha opción. El escaneo de un código QR implicará, en caso de que sea un objeto, el añadido de ese objeto al inventario. Una vez escaneado el código, la aplicación volverá a la interfaz en la que se encontraba.

Es importante diferenciar entre añadir un objeto mediante código QR que mediante ID (que veremos a continuación) ya que ambas funcionalidades realizan funciones muy parecidas pero con diferencias notables.

ii. Requisito funcional 2: "Añadir objeto por ID"

Se brindará al usuario la posibilidad de añadir objetos al inventario mediante un ID de dos dígitos. Cuando el usuario seleccione tal opción se le abrirá un teclado numérico mediante el cual podrá introducir el ID. Una vez introducido el ID se

comprobará si coincide con un objeto existente y, en caso afirmativo la aplicación volverá a la interfaz en la que se encontraba antes de seleccionar añadir un objeto por ID.

En caso de que el usuario se equivoque introduciendo el id se le notificará y se volverá a pedir el ID.

Esta diferenciación entre añadido de objeto por ID y por código QR se describe en la sección [¿Dos maneras de hacer lo mismo? Códigos qr e id].

iii. **Requisito funcional 3: "Eliminar objeto"**

El usuario podrá eliminar los objetos que tenga en su inventario si lo considera oportuno. Esta característica puede llegar a ser útil dado que el inventario deberá tener una capacidad limitada.

Para ello el usuario simplemente seleccionará el objeto y escogerá la opción descartar.

iv. **Requisito funcional 4: "Combinatoria de objetos"**

El usuario podrá combinar los objetos que tenga en su inventario con el fin de crear nuevos objetos. Estas combinaciones se conocen con el nombre de recetas y estarán predefinidas.

Se distinguirán tres tipos de recetas:

- Ofensiva
- Curativa
- Apoyo

El usuario simplemente elegirá una de las tres opciones, sin elegir los ingredientes (objetos necesarios para la creación) y la aplicación buscará entre los objetos del inventario para comprobar que tiene los ingredientes necesarios para crear un objeto.

Se detallan los mecanismos para la creación de objetos en la sección [Objetos simples y complejos. Combinatoria de objetos.].

4. **Gestión de las habilidades del jugador**
   a. **Introducción / Propósito de esta característica**
      Esta característica representa la posibilidad del usuario de asignar puntos de habilidad.

b. Secuencia de estímulos / respuestas

El usuario selecciona la habilidad en la cual desea añadir un punto.

El punto de habilidad de añade y el sistema realizará las acciones futuras teniendo en cuenta este nuevo valor.

c. Requisitos funcionales asociados.

i. Requisito funcional 1: "Añadir punto de habilidad"

El usuario podrá seleccionar una habilidad para la cual quiera añadir un punto de habilidad. Para esto deberá tener un punto de habilidad disponible.

REQUISITOS NO FUNCIONALES

ORGANIZACIÓN POR TIPOS

USABILIDAD

- Interfaz sencilla. Los botones han de ser grandes de manera a que el usuario no tenga que "apuntar" para pulsarlos.
- La interfaz debe mantener un esqueleto fijo. Este requisito es puramente estético. El fin perseguido por el mismo es dar la sensación de que la aplicación (en la parte del juego) se corresponde con un dispositivo físico con botones reales, que se mantienen siempre en el mismo lugar.

INTEGRACIÓN CON EL ENTORNO FÍSICO

- La aplicación no deberá eclipsar al juego de mesa en ningún aspecto, dejando la mayor parte del protagonismo al mismo.
- La integración con el mismo debe resultar intuitiva.

IMPLEMENTACIÓN

- La aplicación se implementará en un lenguaje de programación a objetos, en concreto JAVA.

MANTENIBILIDAD Y PORTABILIDAD

- La aplicación deberá estar disponible para Android para, como mínimo, la versión de API 14.

SEGURIDAD

- En caso de incorporarse datos sensibles de los usuarios, se encriptarán.

- Para usar la aplicación será necesario identificarse. Los datos de un jugador no estarán accesibles a nadie más que al propio jugador.

CASOS DE USO

DESCRIPCIÓN DE LOS CASOS DE USO

CASO DE USO: REGISTRAR USUARIO

1. **Actor primario**

   Usuario no registrado.

2. **Interesados y objetivos**

   Un usuario no registrado quiere registrarse en el sistema.

3. **Precondiciones**

   El usuario no puede estar registrado en el sistema.

4. **Garantía de éxito (Post-condiciones)**

   El usuario es registrado en el sistema, se autentica inmediatamente y accede al menú principal de la App.

5. **Escenario principal de éxito**
   a. El usuario no registrado inicia la aplicación y selecciona la opción de registro
   b. La aplicación muestra el formulario de registro.
   c. El usuario rellena los campos del formulario.
   d. La aplicación comprueba que el nombre de usuario escogido no está siendo utilizado por otro usuario.
   e. La aplicación comprueba que las contraseñas introducidas por el usuario coinciden
   f. La aplicación autentica al usuario y lo redirige al menú principal de la app
6. **Extensiones (Flujos alternativos)**
   a. El usuario ha introducido un nombre de usuario en uso. La aplicación no realiza el registro y vuelve a solicitar los datos al usuario.
   b. Las contraseñas introducidas por el usuario no coinciden. La aplicación no realiza el registro y vuelve a solicitar los datos al usuario.
   c. En cualquier momento la aplicación falla y se cierra.
7. **Requisitos especiales**

8. **Lista de variaciones de tecnología y datos**
9. **Frecuencia de ocurrencia**

   Una vez por usuario que use la aplicación.

10. **Temas abiertos**

CASO DE USO: AUTENTICARSE EN EL SISTEMA

1. **Actor primario**

   Usuario registrado.

2. **Interesados y objetivos**

   Un usuario registrado desea autenticarse en el sistema.

3. **Precondiciones**

   El usuario debe estar registrado en el sistema.

4. **Garantía de éxito (Post-condiciones)**

   El usuario se autentica en el sistema con éxito y la aplicación le dirige a la pantalla principal de la App.

5. **Escenario principal de éxito**
   a. El usuario inicial la aplicación y se le muestra el formulario de inicio de sesión.
   b. El usuario introduce sus credenciales de inicio de sesión.
   c. La aplicación comprueba el nombre de usuario y la contraseña del usuario
   d. La aplicación autentica al usuario en el sistema.
   e. Se muestra la pantalla principal de la App.
6. **Extensiones (Flujos alternativos)**
   a. El usuario introduce credenciales erróneas. La aplicación vuelve a solicitar las credenciales al usuario.
   b. En cualquier momento la aplicación falla y se cierra.
7. **Requisitos especiales**
8. **Lista de variaciones y de tecnologías y datos**
9. **Frecuencia de ocurrencia**

   A cada vez que el usuario inicia la aplicación, esto es, aproximadamente unas veces al mes.

10. **Temas abiertos**

CASO DE USO: AÑADIR OBJETO AL INVENTARIO MEDIANTE CÓDIGO QR

1. **Actor primario**

   Usuario autenticado con una partida iniciada.

2. **Interesados y objetivos**

   El usuario desea añadir un objeto al inventario.

3. **Precondiciones**

   El usuario debe estar autenticado y debe tener una partida iniciada.

   El usuario debe tener acceso al botón “scan”.

4. **Garantía de éxito (Post-condiciones)**

   El objeto es añadido con éxito al inventario del usuario, la aplicación vuelve a la pantalla en la que el usuario se encontraba antes de pulsar el botón “scan”.

5. **Escenario principal de éxito**
   a. El usuario pulsa el botón de scan.
   b. Se inicia el escáner de códigos QR.
   c. El usuario escanea un código QR.
   d. Se cierra el escáner de códigos QR.
   e. La aplicación vuelve a la pantalla en la que se encontraba el usuario.
   f. La aplicación realiza una búsqueda en base de datos con el código leído.
   g. La aplicación comprueba que el objeto de base de datos devuelto por la consulta es efectivamente un objeto que se puede añadir al inventario.
   h. La aplicación añade el objeto al inventario.
   i. La aplicación informa al usuario de que se ha añadido el objeto al inventario, especificando el nombre del mismo.
6. **Extensiones (Flujos alternativos)**
   a. El código escaneado no se corresponde con un objeto. La aplicación no lo añade al inventario.
   b. En cualquier momento la aplicación falla y se cierra inesperadamente.
7. **Requisitos especiales**
8. **Lista de variaciones y de tecnologías y datos**
9. **Frecuencia de ocurrencia**
   a. Del orden de 15-20 veces por partida.
10. **Temas abiertos**

CASO DE USO: COMBINAR OBJETO

11. Actor primario

Un usuario autenticado, con una partida iniciada y con objetos en el inventario.

12. Interesados y objetivos

El usuario desea combinar objetos para crear un objeto nuevo.

13. Precondiciones

El usuario debe tener objetos correspondientes a una receta en su inventario y encontrarse en la pantalla de inventario.

14. Garantía de éxito (Post-condiciones)

La aplicación crea el objeto combinado, lo añade al inventario y borra los objetos usados en el proceso de creación del inventario.

15. Escenario principal de éxito

a. El usuario selecciona un tipo de creación (Ofensiva, curativa o de apoyo).
b. La aplicación obtiene todas las posibles recetas de la base de datos.
c. La aplicación comprueba si el usuario tiene alguna combinación de objetos que satisfaga una receta.
d. En caso afirmativo, la aplicación añade al inventario del usuario el nuevo objeto.
e. La aplicación suprime del inventario los objetos usados en la creación del objeto.
f. La aplicación avisa al usuario de que se ha creado un objeto, indicando el nombre del mismo.

16. Extensiones (Flujos alternativos)

a. El usuario no dispone de los objetos necesarios para la creación de un objeto del tipo seleccionado por el usuario. La aplicación notifica al usuario de este hecho.
b. En cualquier momento la aplicación falla y se cierra inesperadamente.

17. Requisitos especiales

18. Lista de variaciones y de tecnologías y datos

19. Frecuencia de ocurrencia

a. Del orden de 10 veces por partida.

20. Temas abiertos

DISEÑO DE LA SOLUCIÓN

A continuación expondremos el diseño de la solución empezando por los diagramas de clase.

El patrón de diseño escogido es MVC (Modelo – Vista – Controlador). Vamos a hacer un breve repaso sobre el patrón MVC.

MODELO – VISTA – CONTROLADOR

El Modelo – Vista – Controlador (MVC) es un patrón de diseño de arquitectura de software que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones.

Expresado de manera más entendible, esto significa que el MVC separa la interfaz de usuario de los datos persistentes de la aplicación y utiliza una serie de controladores para realizar las acciones pertinentes.

En MVC el modelo representa los datos, la vista la interfaz de usuario y el controlador es el encargado de conectar interfaz y datos.

El modelo, aparte de representar los datos, es el encargado de gestionar los mismos; Esto significa que las manipulaciones de los datos deberá hacerlas el modelo mediante una serie de métodos que estén definidos en él mismo.

El controlador es el encargado de responder a los eventos de entrada del usuario. Cuando sea preciso, enviará peticiones al modelo para modificar u obtener datos.

La vista es básicamente la interfaz de usuario. Presenta los datos del modelo en un formato reconocible por el usuario. Todas las interactuaciones que efectúe el usuario con la interfaz serán traducidas a peticiones a los controladores.

En la [**Error! Reference source not found.**] podemos ver un pequeño esquema de cómo funciona el patrón MVC

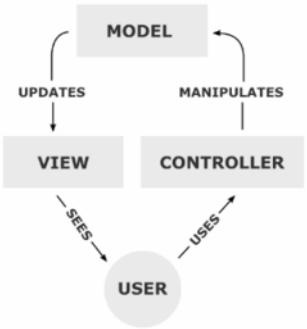

ILUSTRACIÓN 7: ESQUEMA MVC [24]

En esta sección estudiaremos el diseño de clases de la aplicación. Por motivos obvios de facilidad de lectura y explicación, separaremos el diagrama principal y haremos una profundización por paquetes.

Para empezar pondremos una visión global, esto es [Ilustración 8: Diagrama de clases completo]

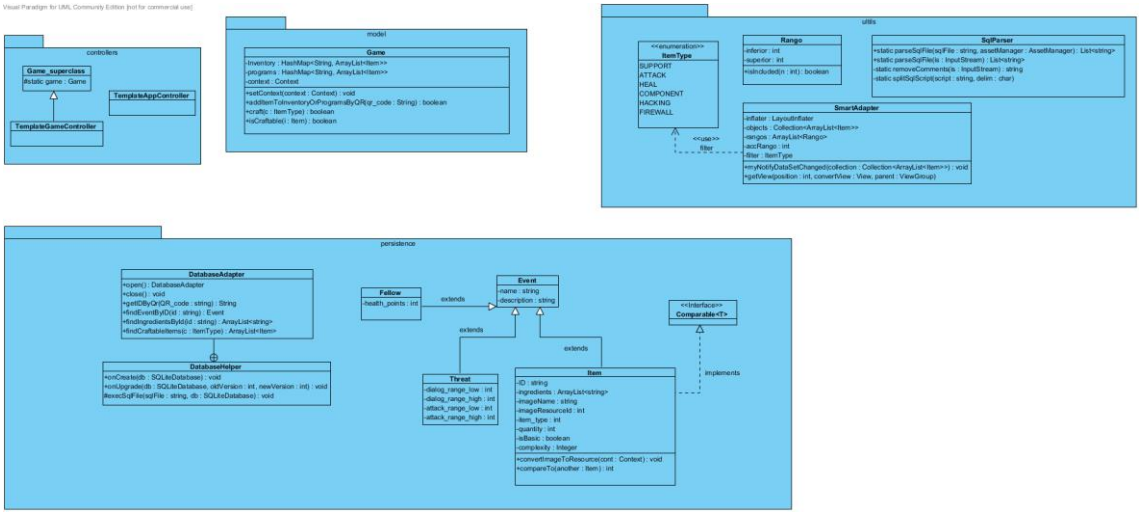

ILUSTRACIÓN 8: DIAGRAMA DE CLASES COMPLETO

PAQUETE CONTROLLER

A continuación nos centraremos en el paquete “controllers”, que recoge los controladores de la aplicación.

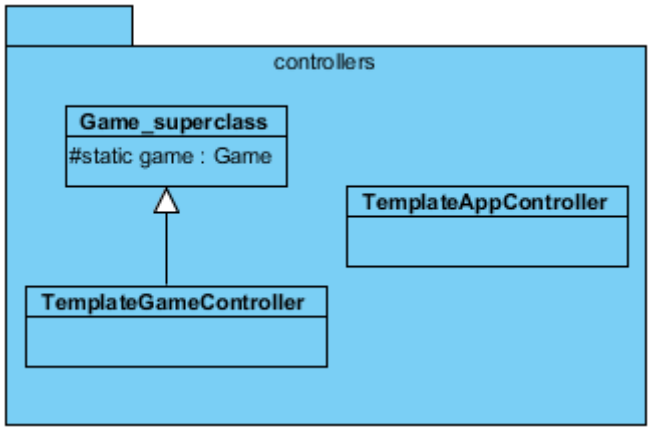

ILUSTRACIÓN 9: PAQUETE CONTROLLERS

Este paquete es deliberadamente sencillo, ya que sólo es una representación. Poner todos los controladores en el diagrama daría lugar a un diagrama excesivamente

sobrecargado. Todos los controladores siguen el mismo esquema que vamos a explicar a continuación.

Se han dividido los controladores en dos tipos: aquellos que gestionan las interactuaciones del usuario en el contexto de la **App** y aquellos que gestionan las peticiones en el contexto del **Juego**.

Como se puede ver en la [Ilustración 9: Paquete controllers], todos los controladores del juego heredan de una clase llamada "Game_superclass". Esta clase tiene un atributo **estático** de tipo Game (Clase que veremos más adelante). Esto es necesario debido a que mientras el usuario va navegando por la parte del juego de la aplicación tendrá una "**sesión de juego**" iniciada. Esta información sobre la partida debe conservarse, esto explica este atributo.

En caso de que el lector no entienda las implicaciones que tiene un atributo estático vamos a esclarecerlo rápidamente.

Un atributo estático sólo puede tener **una instancia** a la vez. Esto significa que para cada instancia de un controlador, que son muchos, sólo habrá un atributo juego compartido por todos. Esto implica que todos los controladores tendrán la misma información de partida, guardada en la instanciación de la clase Game. De esta forma no es preciso que los controladores se pasen esta información mediante "Intents" de Android.

Los controladores de la App no precisan de una información de este tipo luego no heredan de esta clase.

PAQUETE MODEL

Aunque este paquete se llama “model”, no engloba el concepto de modelo explicado anteriormente. Este concepto es englobado por una combinación del paquete model y el paquete persistence, que veremos más adelante

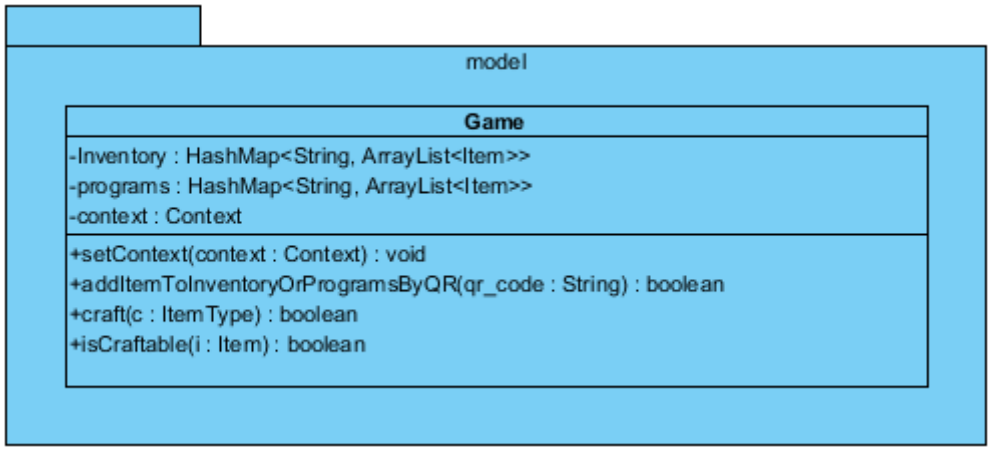

ILUSTRACIÓN 10: PAQUETE MODEL

Este paquete consta de una única clase: **Game.** Esta clase contiene la información de la partida de un jugador mientras está en la parte de juego de la aplicación.

Recordaremos rápidamente que la “parte de juego” de la aplicación se refiere a las interfaces y mecanismos subyacentes a la instancia del juego. Es decir, cuando un jugador inicia una partida, entrará en el menú del juego y podrá realizar operaciones directamente relacionadas con el juego de mesa. Esto se ha explicado previamente en [Definiciones, acrónimos y abreviaturas] en el análisis de requisitos.

En esta clase se guardará, pues, toda la información relevante al juego. En este diseño sólo se han considerado las partes de inventario y programas.

Como podemos ver se guarda la información del inventario y de los programas en un HashMap con clave e Item. Esta decisión se explica más adelante en la sección de [Implementación].

El contexto se refiere al contexto de Android, es decir, el contexto de ejecución de la aplicación. Es necesario guardar esta información para poder acceder en algunos casos a ciertos recursos de la aplicación.

Los métodos de la clase Game se explican prácticamente por sí solos. Con estos métodos podemos añadir objetos al inventario mediante un código QR, combinar objetos (craft) y comprobar si un objeto se puede construir.

PAQUETE PERSISTENCE

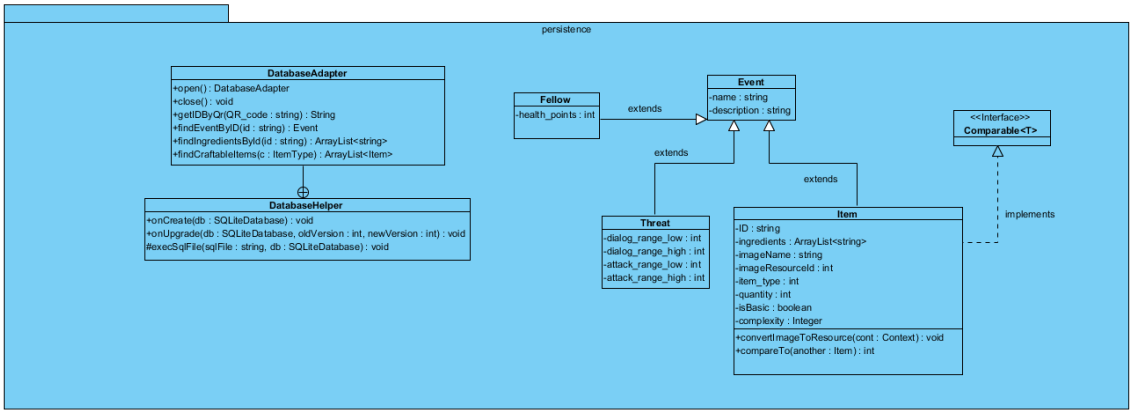

ILUSTRACIÓN 11: PAQUETE PERSISTENCE

Debido a que este paquete vuelve a ser una vez más demasiado grande, vamos a subdividirlo en dos.

DATABASEADAPTER

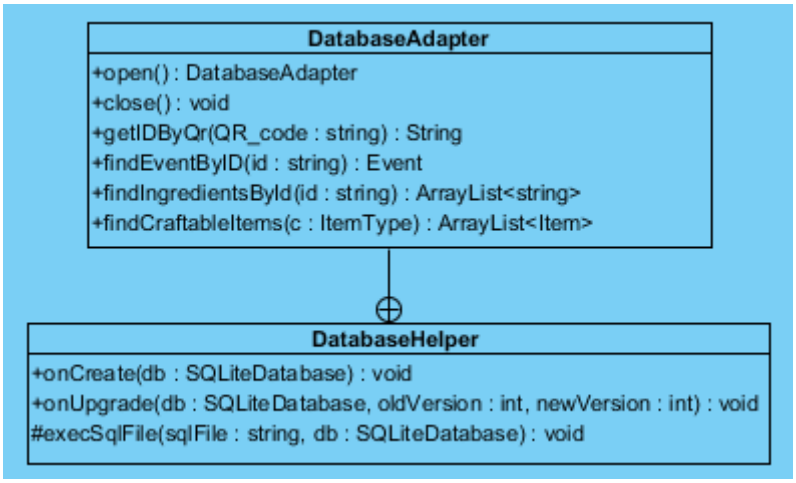

ILUSTRACIÓN 12: DATABASEADAPTER

La clase DatabaseAdapter es la encargada de la comunicación directa con la base de datos de la aplicación. Contiene los métodos de búsqueda de información pertinentes.

La clase DatabaseHelper es una clase privada contenida dentro de la clase DatabaseAdapter. Esta clase es la encargada de inicializar la base de datos y actualizarla en caso de que se cambie de versión de la misma. La base de datos, aunque no es el propósito de este documento, se carga a través de unos ficheros mediante el método execSqlFile.

ABSTRACCIÓN DE LA BASE DE DATOS

A continuación vamos a ver las clases que abstraen la base de datos.

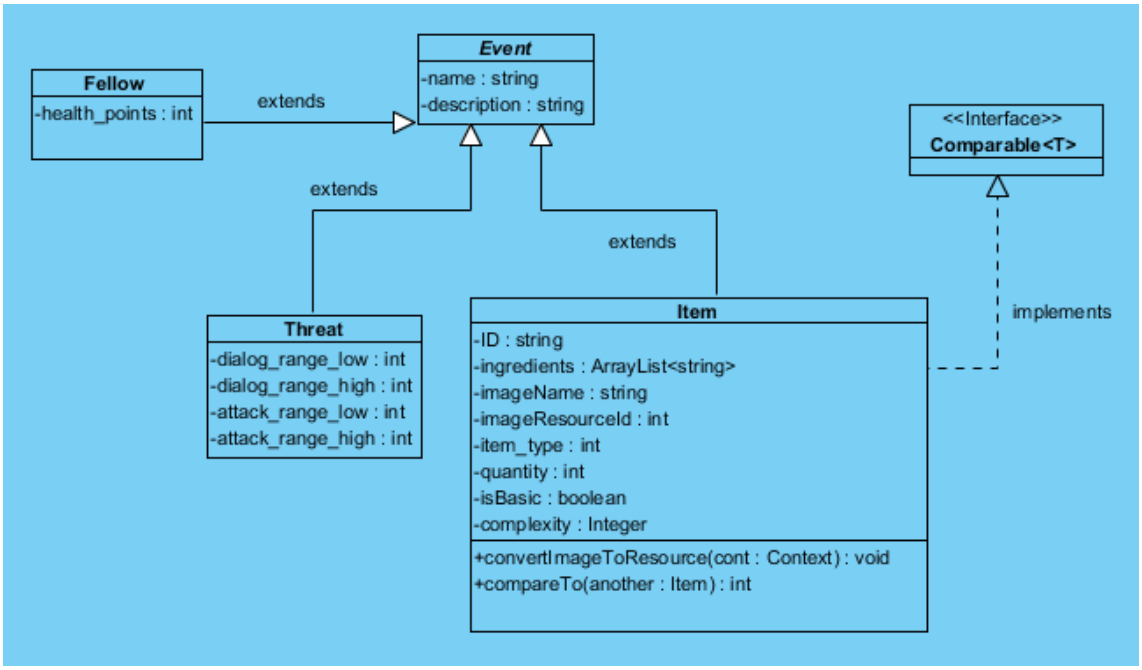

ILUSTRACIÓN 13: ABSTRACCIÓN DE LA BASE DE DATOS

Estas clases son una representación Java de la información contenida en base de datos. Se han organizado dicha información de la siguiente forma:

- **Event**: Esta clase abstracta reúne los elementos comunes de todos los eventos de la base de datos. En el contexto de este proyecto, un evento puede ser un **objeto**, un **compañero** o una **amenaza**. Para entender estos eventos es preciso pensar en el juego de mesa; Un objeto se corresponde a una carta de objeto simple o a un objeto combinado, una amenaza se corresponde a una carta de enemigo o a una amenaza del entorno para el jugador y un compañero se corresponde a una carta de compañero del juego de mesa. Estos conceptos que pueden resultar un poco confusos se esclarecerán más adelante.
- **Fellow**: Esta clase representa a un compañero en el juego de mesa, en esta versión sólo tiene información sobre los puntos de vida del mismo.
- **Amenaza**: Esta clase recoge la información de un posible enemigo. Como puede ver el lector, contiene un rango de ataque y un rango de diálogo. Estos dos conceptos, ataque y diálogo, serán aclarados en el [Anexo A: Diseño del sistema de conflictos]

- **Item**: Esta clase abstrae el concepto de objeto. Contiene toda la información de un objeto.
    - El atributo **ID** contiene el identificador único del objeto. Este identificador será el que se use como campo clave en el HashMap del inventario, aunque eso se detallará más adelante.
    - **Ingredients** contiene identificadores de objetos necesarios para la creación del objeto. Podría pensarse que este atributo es inútil ya que si el objeto está instanciado, es que ha sido creado luego no tiene sentido mantener esta información. Más adelante en la sección de [Implementación] veremos que resulta más conveniente tener esta información.
    - **imageName** contiene el nombre que tiene la imagen representativa en el sistema de archivos. Está información será usada en un futuro para sacar el identificador asignado por Android a esta imagen en tiempo de ejecución a través del contexto de la aplicación.
    - **imageResourceId** contiene el identificador previamente nombrado. A la hora de mostrar la imagen, es preciso señalar a Android a qué imagen nos referimos. Todos los recursos de una aplicación Android tiene un identificador asignado. Mediante este identificador conseguimos solicitar la imagen adecuada para mostrar.
    - **item_type** contiene información acerca de si el objetivo es de tipo ofensivo, curativo o de apoyo.
    - El atributo **quantity** se explica por sí mismo: indica la cantidad de ese objeto en el inventario. Si esto parece extremadamente simple, es en realidad una solución a un problema de diseño encontrado. Este problema y su solución se discuten en el apartado [Un problema visual: SmartAdapter]
    - El atributo booleano **isBasic** informa de si el objeto es simple o complejo.
    - **complexity** contiene la información de complejidad del objeto. Esta complejidad se calcula con respecto al número de objetos contenidos en ingredients. Los objetos simples tiene complejidad 0.

    Es importante destacar que la clase Item implementa la interfaz **Comparable**. Esto se debe a que los objetos del inventario son ordenados por complejidad. Esto se lleva a cabo con ayuda del método compareTo, como ya supondrá el lector.

PAQUETE UTILS

El paquete Utils contiene todas las funciones auxiliares que puedan hacer falta para el funcionamiento de la aplicación.

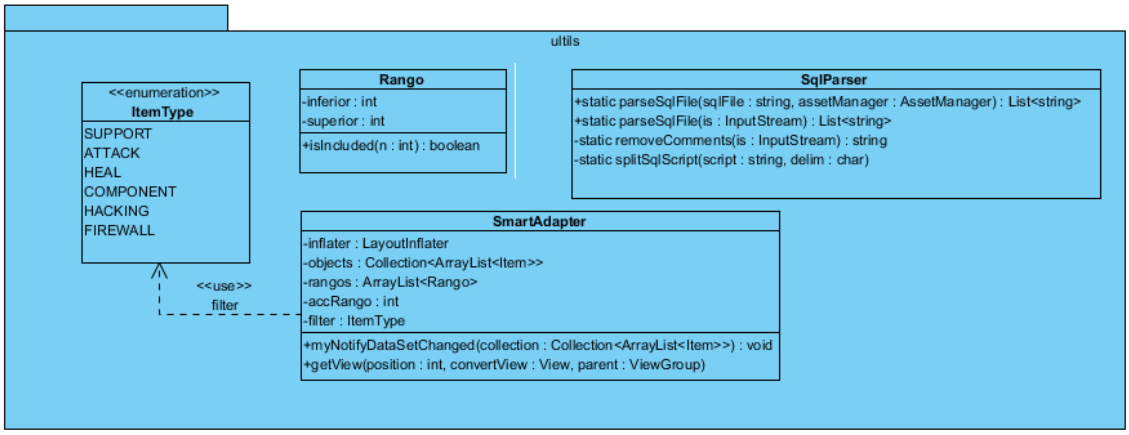

ILUSTRACIÓN 14: PAQUETE UTILS

Sólo vamos a hacer un breve acercamiento al paquete utils en esta sección.

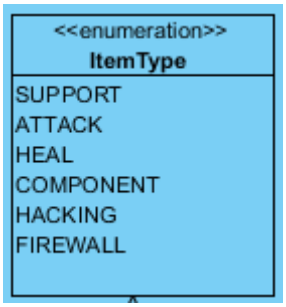

ILUSTRACIÓN 15: ENUMERACIÓN ITEMTYPE

La enumeración ItemType no es más que una representación de los posibles tipos de objeto. El fin de esta clase es hacer el código más comprensible y organizado en las comparaciones de tipos de objeto.

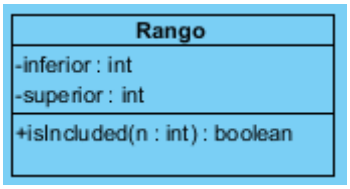

ILUSTRACIÓN 16: CLASE RANGO

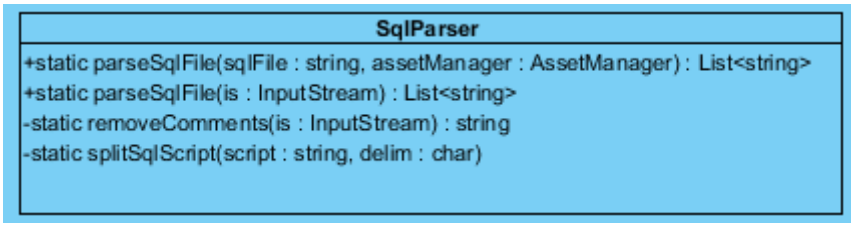

ILUSTRACIÓN 17: CLASE SQLPARSER

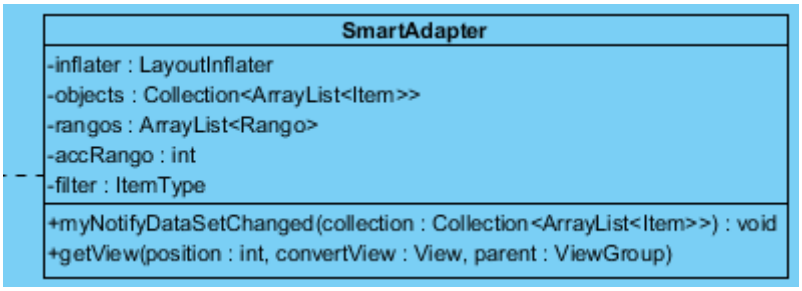

ILUSTRACIÓN 18: CLASE SMARTADAPTER

De estas tres últimas clases sólo nos fijaremos en SmartAdapter, aunque no en esta sección. La clase SqlParser, aunque interesante, corresponde a operaciones con los ficheros de la base de datos y estos no son el objeto de este proyecto. La clase Rango existe exclusivamente por motivos de organización del código y no merece nuestra atención.

Con esto terminamos nuestro análisis del diagrama de clases.

Durante el proceso de desarrollo y diseño de este proyecto, se han encontrado numerosos problemas de diseño que, aunque no tan técnicos, si importante por su repercusión en el diseño final de la aplicación. Para solucionarlos, se han realizado numerosas reuniones con el diseñador del juego de mesa y la productora. A continuación vamos a ver algunos de estos problemas.

EQUILIBRIO ENTRE JUEGO DE MESA Y APLICACIÓN

Un concepto importante del que hemos hablado anteriormente es la necesidad de que el producto final sea un simbiosis entre juego de mesa y aplicación, sin prevalencia clara de ninguno de los dos aspectos.

En varias reuniones con la productora y el diseñador de la parte del juego de mesa se han discutido numerosos puntos relacionados con este tema.

EL PROBLEMA INICIAL: DEMASIADA INTRUSIÓN DE LA APLICACIÓN

Un notable problema que se ha encontrado durante el diseño del juego es que gran parte de las soluciones se resolvían mediante la aplicación. Por ejemplo en un principio se votó por eliminar las cartas de objeto físicas del juego de mesa y gestionar los objetos exclusivamente con la aplicación; También se abogó por introducir un dado en la aplicación y eliminar el sistema de dado tradicional.

Estos enfoques han requerido una labor de ingeniería para dejar más aspectos del juego en el mundo físico.

EFECTOS SECUNDARIOS: INTEGRACIÓN DE LA APLICACIÓN CON EL ENTORNO FÍSICO

Las soluciones al problema inicial, aunque necesarias, traían con ellas múltiples problemas de integración de la aplicación con el entorno físico del juego de mesa.

Retomando los ejemplos anteriores, era necesario que los resultados del dado fueran comunicados a la aplicación para la resolución de conflictos (La implementación de los conflictos es considerada de fase 2 del proyecto. Sin embargo ya se han diseñado las soluciones a implementar y, por ello, se detallarán), también se convirtió en una necesidad conseguir comunicar a la aplicación cuándo un jugador obtiene una carta de objeto física para que la pudiese introducir en el inventario.

Las soluciones aportadas se corresponden más a la parte de [Implementación], luego se detallarán en dicha sección

IMPLEMENTACIÓN

En esta sección detallaremos las implementaciones aportadas como soluciones a los problemas de diseño previamente descritos, así como problemas de implementación en sí.

Obviaremos las implementaciones de funcionalidades como el registro de usuario, el inicio de sesión, la navegación de menús, etc., y nos centraremos en los algoritmos y problemas de implementación más interesante.

OBJETOS: UNA SOLUCIÓN HÍBRIDA

En toda esta sección nos referiremos a instancias de la clase java Item como “objetos”.

Como ya se ha explicado, en un enfoque inicial se pretendía integrar toda la funcionalidad de los objetos en la aplicación, sin embargo se descartó rápidamente para dar más protagonismo al juego de mesa.

La solución tomada para este problema consiste en tener una mezcla entre el juego de mesa y la aplicación. Concretamente, una serie de objetos, a los que llamaremos “**simples**”, existen físicamente bajo forma de carta. Otros objetos, a los que llamaremos “**complejos**” sólo existen en el ámbito virtual de la aplicación.

Durante una partida un jugador va cogiendo cartas de objetos simples para añadirlos a su inventario “físico”. Inmediatamente surge un problema: ¿Cómo comunicamos a la aplicación que el jugador ha cogido tal carta?

La solución aportada a este nuevo problema se basa en dos conceptos: Códigos QR e identificadores (ID).

¿DOS MANERAS DE HACER LO MISMO? CÓDIGOS QR E ID

Ambas funcionalidades, **QR** e **ID**, sirven al mismo propósito: comunicar a la aplicación que un jugador ha tomado una carta de objeto.

¿Por qué dos maneras de hacer lo mismo? La respuesta a esta pregunta es más simple de lo que puede parecer: la lectura de códigos QR con un dispositivo Android es un proceso muy dependiente del entorno. Dependiendo de las condiciones de luz, calidad de la cámara, velocidad del autoenfoque, etc., los tiempos de lectura varían enormemente pudiendo llegar a tardar ¡hasta quince segundos!

Un tiempo de espera tan variable y, en ocasiones, largo no es admisible teniendo en cuenta que la operación de añadir un objeto al inventario es algo que se realizará varias veces por partida. Recordemos que el objetivo de la aplicación es reducir los

tiempos muertos en una partida del juego de mesa, no estorbar a los jugadores. Imagínese que molestia si el jugador activo pasara 15 segundos intentando escanear un objeto que acaba de coger. Probablemente el vecino le diría que lo está haciendo mal y le diría “déjame a mí” ¡ralentizando aún más la partida!

Por ello se introdujo el sistema de añadido de objetos por ID. Dado que el número de objetos en carta física en ningún caso superará un centenar, se ha diseñado un sistema de ID para cada objeto. Cada carta tendrá, en el recto de la misma, un identificador de dos dígitos; Cuando el usuario robe una carta de objeto podrá añadirla a su inventario virtual introduciendo el ID.

¿Eso es todo? ¿Qué sentido tiene mantener el lector de códigos QR entonces? ¡Tiene que existir alguna diferencia!

Efectivamente, la hay. Esta diferencia se basa en un mecanismo de ocultación de la información.

Cuando un usuario añade un objeto por ID, éste se añade al inventario con cierta información oculta.

Esta información oculta es la siguiente:

- Estado del objeto (Durabilidad)
- Información adicional
- Efectividad del objeto

Vamos a aclarar una serie de conceptos sobre los objetos de manera a entender por qué esta diferencia a la hora de añadir objetos puede repercutir en la partida.

Todos los objetos tienen una serie de características asociadas, entre las cuales se encuentran las mencionadas de durabilidad, efectividad e información adicional.

A cada vez que un objeto es usado, la aplicación realiza unos cálculos en función de las características del objeto. Pongamos un ejemplo: si se tiene un objeto de tipo ofensivo, por ejemplo un cuchillo, y se ataca a otro jugador con dicho objeto, el daño que se infligirá al jugador dependerá, entre otros factores, del objeto que se está usando. Se detalla a fondo el sistema de batalla y los factores influyentes en el mismo en la sección [Anexo A: Diseño del sistema de conflictos]

Otro aspecto que se tiene en cuenta cuando se usa un objeto es la durabilidad (o usos) que tiene el objeto. El cuchillo que se ha usado para atacar a otro jugador o a un NPC puede romperse después de haberse usado. Esto viene determinado, como ya supondrá el lector, por la característica de estado del objeto.

Vamos a ver a continuación un concepto clave: tener conocimiento del estado y efectividad de un objeto mejora las mismas; Este concepto, que en principio puede parecer absurdo, se explica muy sencillamente.

Cuando se realiza un uso de un objeto, como ya se ha explicado anteriormente se realizan una serie de cálculos sobre la efectividad y durabilidad del mismo. Si el jugador desconoce estos valores, la aplicación podría asignarlos como le plazca y el jugador no podría hacer nada al respecto. Ahora bien, si el jugador conoce estos valores, la aplicación no podrá hacer tal cosa.

¿Significa esto que al escanear un objeto se están mejorando sus características? En cierto modo sí, veamos cómo es esto.

Como ya supone probablemente el lector, los valores de las características no son asignados arbitrariamente; Estos valores son escogidos dentro de un **rango**. Cada objeto tiene un rango de posibles valores que podrá tomar para ciertas características como la efectividad o la durabilidad.

Cuando un jugador escanea el código QR de un objeto, estos rangos son reducidos, dando lugar a menos posibilidades. ¿Por qué no se escoge un valor directamente en vez de reducir el rango? Esta es una cuestión que abordaremos en breve.

Aclarado este concepto clave de reducción de rangos, de ahora en adelante nos referiremos al escaneo del código QR del recto de la carta (Sí, existen varios códigos QR en una misma carta, pero esta es otra cuestión detallada más adelante en la sección [Exploración del mapa: Un segundo código QR]) como **examinar** un objeto.

Retomando el hilo anterior, cuando un jugador examina un objeto, se reducen los rangos que puede tomar en materia de efectividad y durabilidad, y se muestran al usuario. Si el usuario no examina un objeto, no tendrá idea de los rangos entre los que se mueve su objeto. Esta es una decisión deliberada, no examinar un objeto debe transmitir sensación de inseguridad y aleatoriedad, por ello no se mostrarán estos datos a menos que se examine un objeto.

El hecho de que cuando se examine un objeto se reduzca el rango de posibles valores que puede tomar en vez de suprimir este rango y escoger un valor determinado es para mantener un cierto grado de aleatoriedad cuando se use un objeto. A cada uso se escoge un valor dentro del rango que se usa como factor para resolver la acción en la que está implicada el objeto.

Ni que decir tiene que añadir un objeto por ID no impide que más adelante se examine dicho objeto con el fin de obtener información adicional. Con esto precisamente se consigue que un jugador en *tiempo activo* no se entretenga

intentando leer un código y lo haga en su *tiempo muerto*. Así se solucionan, pues, los problemas encontrados con los tiempos variables en la lectura de código; Si un jugador tarda en leer un código en su *tiempo muerto,* no perturba al resto de los jugadores ni la partida.

Finalmente añadir que la información adicional no repercute en absoluto en la partida, sólo tiene funcionalidad estética y por ello no haremos más que mencionarla.

Con esto terminamos la exposición sobre la diferenciación entre introducir un objeto por ID o por código QR. Resumiendo, examinar un objeto (mediante el código QR) nos devela más información sobre el mismo y nos ayuda en la partida.

EXPLORACIÓN DEL MAPA: UN SEGUNDO CÓDIGO QR

Antes nos hemos referido al hecho de que en una carta existen dos códigos QR en cada carta. ¿A qué se debe esto? La respuesta reside en el concepto de exploración del mapa.

Cuando un jugador está en una casilla (o nodo) del mapa, puede desear robar una carta del mazo de eventos con el fin de conseguir un objeto; sin embargo esto le puede llevar a encontrarse con enemigos. La exploración permite al usuario “adivinar” lo que le va a tocar. Esto se lleva a cabo con el código QR de la parte trasera de la carta. Este código sólo permitirá al usuario obtener una idea muy vaga del tipo de carta que es.

De este modo un jugador con poca vida podría evitar un conflicto no deseado.

INVENTARIO: UN PROBLEMA TÉCNICO

El inventario se diseñó en principio como una lista de objetos. Esta solución ciertamente parece la más intuitiva y básica y, como ya imaginará el lector, trajo varios problemas con ella.

El primer problema que se detectó, aunque leve, se manifestó como un problema de optimización: con una simple lista de objetos, a cada vez que se deseaba sacar un objeto del inventario ya fuera para eliminarlo o cualquier otra acción, era necesario recorrer toda la lista hasta dar con el objeto. Esta búsqueda lineal tiene una complejidad en el **caso peor de N**, un caso medio de N/2 y un caso mejor de 1 siendo N el número de elementos en la lista.

Si consideramos que cuando se desea combinar un objeto (tema que veremos más adelante en detalle) es necesario comprobar si existen determinados objetos en el inventario, nos damos cuenta de que es necesario una mejora en la búsqueda de tales objetos.

Inmediatamente surgió la solución de realizar un mapeo clave-valor para cada objeto, **HashMap** nos permite realizar tal mapeo de manera muy sencilla.

Un dato muy importante es que HashMap provee **tiempos de acceso constantes** [15] para las operaciones básicas como put, get, etc., siendo su complejidad **O(1)**. Esto es una mejora considerable a la búsqueda anterior con complejidad **O(N)**. HashMap, además, incorpora métodos como "contains" que nos permiten comprobar si el objeto que buscamos existe en el HashMap de manera limpia y rápida. Con esto se solucionaban dos problemas: los tiempos de acceso y búsqueda y el comprobar la existencia de un objeto en el inventario.

	Búsqueda lineal	HashMap
Caso mejor	1	1
Caso medio	N/2	1
Caso peor	N	1

TABLA 1: COMPLEJIDAD LINEAL VS HASHMAP

Sin embargo surgió un nuevo problema con el HashMap: la imposibilidad de introducir más de un objeto del mismo tipo. Dado que la clave del HashMap es el ID del objeto, y que no existen varias definiciones de un mismo objeto con distintos IDs (eso sería un despropósito), resultaba imposible introducir más de una instancia de un objeto en el inventario.

Obviamente esto es posible en el entorno del juego; Un jugador es perfectamente capaz de robar dos cartas de "madera" y querrá añadir ambas a su inventario. ¿Cómo se solucionó esto pues? Muy sencillo: con un atributo cantidad (**quantity**) en cada objeto que indica las unidades de este objeto que posee el jugador.

Esto puede resultar chocante en el sentido en que, técnicamente, no debería ser un atributo del objeto ya que la clase objeto es una abstracción de la información contenida en base de datos sobre un objeto. Esto es cierto y es un fallo, sin embargo en esta versión de la aplicación es la solución implementada.

Este atributo de cantidad generó otro problema más: los objetos que tienen características de rango como la efectividad y la durabilidad deberán tener cada uno su rango, no es posible agruparlos en una misma instancia.

Por ello se dio otra vuelta de tuerca al inventario, modificando su estructura. El inventario quedó entonces como un HashMap que realizaba un mapeo de ID de objeto con un ArrayList de objetos.

INSERCIÓN DE OBJETOS EN EL INVENTARIO

En la primera versión, implementada bajo forma de lista simple de objetos, las instancias de los objetos se añadían a la lista sin ningún tipo de comprobación.

En la segunda versión, implementada con un HashMap de clave y objeto los objetos se insertaban exclusivamente si no existía ya una clave cuyo valor se correspondía con el ID del objeto. En caso de existir tal clave, se accedía al objeto guardado en el inventario y se le sumaba uno a la cantidad.

En la tercera versión, que es la implementada actualmente, se realiza la misma comprobación de clave que en la versión 2, sin embargo ahora también se comprueba si el objeto a insertar es complejo o básico, siendo complejo un objeto resultado de una combinación y básico lo contrario. Los objetos básicos no tienen atributos de rango relevantes como los complejos.

Si el objeto a insertar es básico, se suma uno a la cantidad. Si el objeto a insertar es complejo, se inserta la instancia del objeto en el ArrayList de objetos.

El lector agudo se preguntará qué sentido tiene mantener el atributo cantidad. ¿No sería posible insertar las instancias de los objetos sin importar que sean básicos o no? Además así se podría eliminar el atributo quantity que viola la abstracción de la base de datos. Sí, es perfectamente posible. Sin embargo resulta en un desperdicio de memoria guardar varias de un mismo objeto que no se diferencie en nada. Por ello se abogó por mantener el atributo quantity y sólo una instancia de objeto básico a cada vez.

UN PROBLEMA VISUAL: SMARTADAPTER

A continuación vamos a ver muy brevemente un problema de interfaz que surgió a raíz de las modificaciones constantes del inventario. Nuestra exposición será deliberadamente breve dado que la interfaz no es el objetivo de este proyecto, sin embargo este problema de interfaz está fuertemente ligado a la implementación del inventario, con lo cual es justificable contemplarlo a grandes rasgos.

El problema consiste en representar visualmente el inventario. Sabiendo que el inventario puede contener objetos básicos y objetos combinados, cada uno guardado bajo una combinación de una única instancia y el atributo quantity o bien por varias instancias, resulta complicado representar esto visualmente.

La solución aparece bajo forma del SmartAdapter, un adaptador de array para ListView. Resumiendo mucho, este adaptador recibe el inventario, que ha sido convertido a una Colección de ArrayList de objetos. Comprueba el tamaño de cada

ArrayList y lo suma a una variable que acumula el número real de todas las instancias de objetos en el inventario (recordemos que los objetos simples cuentan como una única instancia).

Con este nuevo tamaño total e implementado el método getCount de la interfaz BaseAdapter, conseguimos "engañar" al adaptador de manera a que sea llamado tantas veces como instancias de objetos haya. Una vez hecho esto, ha sido necesario implementar un método de traducción de las posiciones a las posiciones dentro de cada ArrayList. Para explicar esta traducción vamos a ver un breve ejemplo: si el primer objeto tiene dos instancias en su ArrayList y el segundo objeto tiene tres instancias, la posición 4 corresponderá al segundo objeto del segundo ArrayList (comenzando a contar desde 1).

En la [Ilustración 19: Traducciones SmartAdapter] se muestra una simple representación del concepto de traducción. Las traducciones se muestran en las celdas. Ni que decir tiene que se ha de restar 1 a la traducción cuando se acceda al ArrayList para solicitar el objeto por índice.

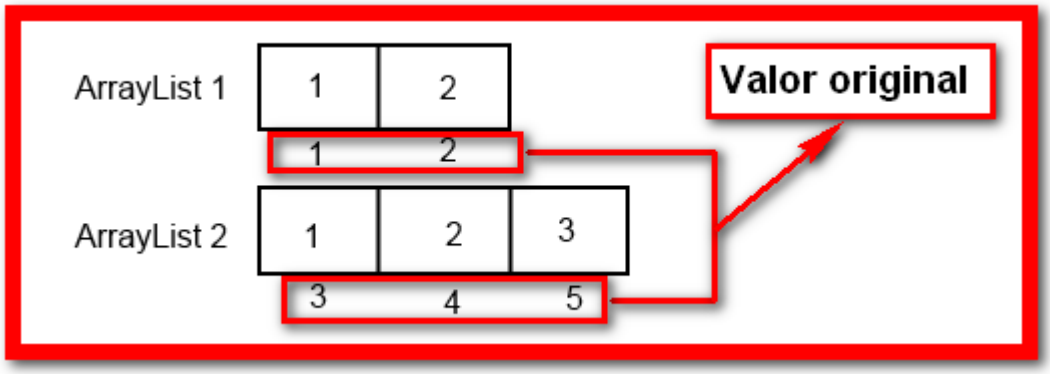

ILUSTRACIÓN 19: TRADUCCIONES SMARTADAPTER

Con este sistema conseguimos acceder a todos los objetos instanciados una vez. Ahora surge el problema de cómo representar los objetos básicos cuto campo quantity tiene relevancia. Esto no nos incumbe ya que no es el objetivo de este proyecto. Sólo diremos que existen varias vistas posibles para la representación de los objetos y en función de si el objeto es básico o no, se mostrará de una forma u otra.

OBJETOS SIMPLES Y COMPLEJOS. COMBINATORIA DE OBJETOS.

Antes hemos mencionado que existen dos tipos de objetos: **simples** y **complejos**.

Como hemos explicado, los simples tendrán representación física y los complejos con competencia exclusiva de la aplicación.

Pero ¿qué son los objetos complejos? La respuesta es tan intuitiva que quizás ya la haya averiguado: un objeto complejo se compone de varios objetos simples.

Evidentemente, no cualquier combinación de objetos simples resulta en un objeto complejo, las posibles combinaciones de objetos están guardadas en base de datos y se conocen como **recetas**.

Vamos ahora a profundizar en cómo se realiza la combinación de objetos.

LA IDEA INICIAL: COMBINACIÓN MANUAL

El primer diseño enfocaba la creación de objetos como un proceso manual en el que el usuario seleccionaba los distintos objetos que deseaba combinar e intentaba crear dicho objeto. En el caso en que el usuario hubiese acertado una combinación existente en una receta, se crearía el objeto.

Sin embargo este enfoque presentaba un serio problema: ¿Tendría cada jugador que aprenderse todas las recetas? ¿Se proporcionaría una lista con las recetas?

Además, ¿qué ocurriría con los objetos implicados en una creación fallida? ¿Quedarían inutilizables en dicho intento frustrado?

Dado que tener a los jugadores consultando constantemente una lista de recetas para comprobar si podían crear un objeto se nos antojaba como una idea espantosa en términos de jugabilidad y forzar a los jugadores a aprenderse una lista es sencillamente una idea absurda, se abogó por investigar algún otro sistema. Es así como surgió la creación automatizada de objetos divididos por categoría.

UN SEGUNO ENFOQUE: COMBINACIÓN AUTOMATIZADA DE OBJETOS

La creación automatizada de objetos es un concepto muy simple en la práctica: cuando un usuario desea combinar un objeto simplemente se dirige a su inventario y selecciona qué tipo de objeto quiere crear. Existen tres tipos de objeto:

- Ofensivo
- Curativo
- De apoyo

Al seleccionar una opción, el control pasa por completo a la aplicación. Estos son los pasos que sigue el algoritmo de combinación de objetos:

1. La aplicación extrae todas las recetas de la base de datos que correspondan a objetos cuyo tipo sea el que se desea crear.

2. La aplicación comprueba todas las recetas y marca aquellas que se pueden crear.
3. Si existen recetas cuyas condiciones se satisfagan, pasará a escoger qué objeto crear. Para ello se ha asignado a cada receta un valor de **complejidad**. Dicha complejidad se basa en el número de ingredientes que se necesitan para satisfacer la receta; A mayor número de ingredientes, mayor la complejidad.
   Combinando la complejidad de las recetas y la **habilidad** del jugador para combinar objetos, se escogerá el objeto a crear. En caso de existir varias posibilidades pasados estos filtros, se escogerá aleatoriamente.
   Las habilidades y como repercuten en el juego son estudiadas más adelante en la sección [Habilidades del jugador].
4. Una vez escogido la receta a crear, se guardará una instancia del objeto combinado en el inventario.
5. El algoritmo procederá ahora a eliminar del inventario del jugador los objetos implicados en la creación del objeto combinado. Para ello el algoritmo prestará atención a si el objeto a borrar es simple o complejo. En caso de ser **complejo**, borrará una instancia del objeto del ArrayList asociado al ID del objeto. Si el ArrayList se queda vacío, se borrará esa entrada del HashMap de inventario. En caso de ser un objeto **simple**, restará uno al atributo quantity y si llega a cero se eliminará esa entrada del HashMap de inventario.

Cuando el usuario intenta crear un objeto y no puede, es informado; Cuando se crea un objeto se crea, se informa al usuario especificando el nombre del objeto, tal y como se solicitaba en el Análisis de requisitos.

Esto resume lo que hace el sistema de combinación de objetos automatizado.

Actualmente no existen recetas que contengan algún ingrediente que sea un objeto combinado, sin embargo el sistema está preparado para una posible incorporación de ese tipo. Sólo sería necesario añadir estas recetas a la base de datos.

LECTURA DE CÓDIGOS QR

La lectura de códigos QR hace uso de una librería externa de Google llamada ZXing. Una vez integrada esta librería en nuestra aplicación para hacer uso de esta hemos de seguir un proceso que describiremos a continuación. La integración de una librería externa en nuestro proyecto se detalla en el [

Anexo C: Integración de un lector QR en una aplicación Android]

La manera con la cual hemos hecho uso del lector QR ha sido mediante intents y la función startActivityForResult.

```
Intent intent = new Intent(getApplicationContext(),
        CaptureActivity.class);
intent.setAction("com.google.zxing.client.android.SCAN");
// This stops saving ur barcode in barcode scanner app's history
intent.putExtra("SAVE_HISTORY", false);
startActivityForResult(intent, 0);
```

ILUSTRACIÓN 20: LECTURA DE CÓDIGO MEDIANTE INTENT

Cuando creamos el intent, especificamos la clase de la librería externa que se encarga de leer códigos.

Es importante añadir la actividad correspondiente a este intent en el archivo manifest.xml

```
<activity
    android:name="com.google.zxing.client.android.CaptureActivity"
    android:configChanges="orientation|keyboardHidden"
    android:screenOrientation="landscape"
    android:windowSoftInputMode="stateAlwaysHidden" >
</activity>
```

ILUSTRACIÓN 21: QR SCANNER MANIFEST ACTIVITY

Una vez hecho esto podemos ejecutar con tranquilidad el método startActivityForResult que, básicamente, ejecuta una actividad y cuando termina llama al método onActivityResult. En este método es donde procesaremos los datos de obtenidos del escáner.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {

    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 0) {
        if (resultCode == RESULT_OK) {
            String contents = data.getStringExtra("SCAN_RESULT");
            Log.d(TAG, "contents: " + contents);
            game.addItemToInventoryOrProgramsByQR(contents);
        } else if (resultCode == RESULT_CANCELED) {
            // Handle cancel
            Log.d(TAG, "RESULT_CANCELED");
        }
    }
}
```

ILUSTRACIÓN 22: ONACTIVITYRESULT

La variable data contiene los datos que necesitamos bajo un extra cuyo nombre es "SCAN_RESULT".

Con esto ya tenemos el contenido del código QR descodificado en la variable "contents" y podemos procesarla como nos plazca. En nuestro caso esa variable contendrá una cadena con información que se trata en la función de "addItemToInventoryOrProgramsByQR"

PROGRAMAS: UN SEGUNDO INVENTARIO

La aplicación debe permitir a un usuario añadir un "programa" a la misma. Estos programas tienen un comportamiento prácticamente idéntico al de los objetos (ítems). De hecho son considerados objetos por el programa, la única diferencia que tienen con estos es que deben ser mostrados en una interfaz distinta a la del inventario. Los programas han de ser mostrados en la interfaz de "computing".

La inserción de estos programas sigue exactamente el mismo proceso que el de los objetos en el inventario, luego no perderemos tiempo con ello.

Un programa puede ser "cargado en el sistema" y esto básicamente cambiará una serie de atributos en el juego.

La idea es que, en la fase dos del proyecto, varios jugadores serán capaces de "hackear" dispositivos ajenos para inutilizarlos temporalmente. Para poder realizar esto necesitarán un programa de hacking. También existe la posibilidad de defenderse de posibles ataques enemigos mediante programas de Firewall. Sin embargo todos estos aspectos se implementarán en la fase dos del proyecto. Una descripción más detallada del hacking y de demás aspectos del juego se proporciona en el [Anexo D: Argumento y funcionamiento del juego de mesa].

HABILIDADES DEL JUGADOR

Como ya hemos mencionado , un jugador podrá gestionar sus habilidades a lo largo de la partida. ¿Qué son las habilidades y qué fin tienen? Esto es lo que vamos a intentar explicar en esta sección.

TIPOS DE HABILIDADES

Existen cinco habilidades en el juego:

- Diálogo
- Conflicto
- Informática
- Creación
- Exploración

En la siguiente figura podemos ver una representación gráfica de estas habilidades. Si nos fijamos, vemos que se pueden invertir hasta 3 puntos en cada habilidad.

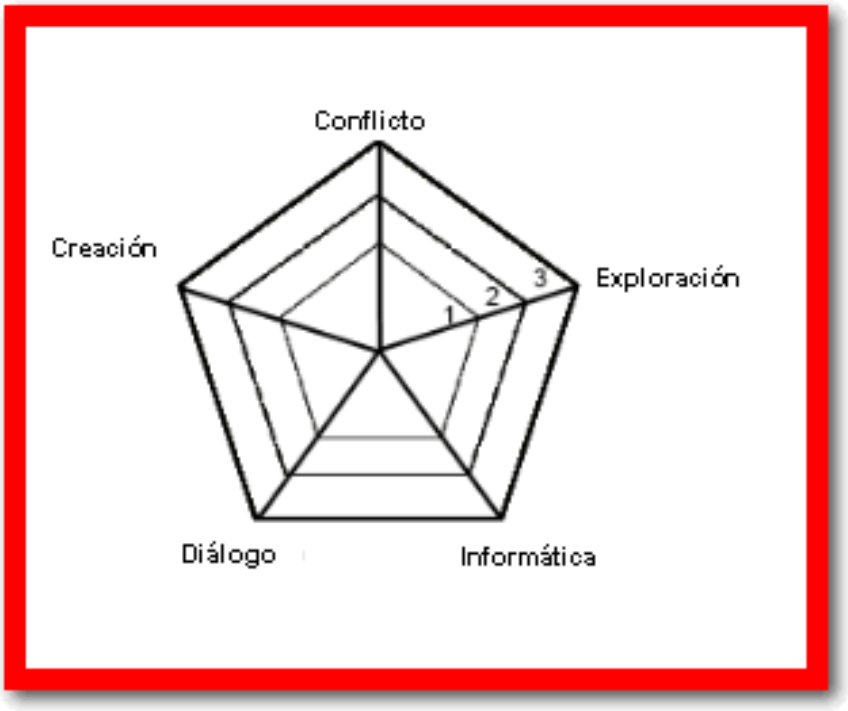

ILUSTRACIÓN 23: HABILIDADES DEL JUGADOR

Es preciso añadir que la representación en pentágono no es una casualidad; Cada habilidad tiene dos habilidades opuestas. Esto implica que una vez alcanzada la maestría en una habilidad no se podrá alcanzar la maestría en una de las habilidades

opuestas. ¿Qué implica alcanzar la maestría? Vamos a ver precisamente esto en la siguiente sección.

NIVELES DE HABILIDAD: MAESTRÍA

Como hemos dicho, existen tres posibles "niveles" de habilidad, alcanzar el tercer nivel de habilidad implica alcanzar lo que denominaremos "maestría".

Cada nivel implica una serie de mejoras acumulativas; alcanzar la maestría implica, además, una mejora adicional. A continuación detallaremos qué mejoras implica cada "subida de nivel", separado por tipo de habilidad.

- Invertir un punto de habilidad en **conflicto** implica una mejora de daño base para los conflictos a cada subida de nivel. Al alcanzar la maestría se realizarán dos intentos de ataque en caso de derrota.
- En el **diálogo** se mejorará la potencia del mismo, aumentando la posibilidad de un diálogo exitoso. Al alcanzar la maestría existirá un segundo intento de diálogo en caso de fracaso.
- En **informática**, subir de nivel implica que los ataques informáticos tendrán mayor probabilidad de éxito además de afectar a un mayor número de áreas del dispositivo enemigo, o el tiempo que duran los efectos. La maestría implica obtener la posibilidad de introducir un "sniffer" en un dispositivo enemigo. El sniffer sólo se podrá usar con un oponente a la vez y permite monitorizar las conversaciones del mismo, obtener información como la vida, el estado del firewall, etc.
- En lo que respecta a la **exploración**, un punto de habilidad mejorará el estado de los objetos que obtenga este jugador (con estado nos referimos a los rangos de efectividad y la durabilidad). Un maestro podrá robar una carta más cuando se explore el terreno.
- Un nivel más de **creación** aumenta el abanico de objetos creables, esto se consigue asignando a cada receta un requisito de nivel. La maestría implica que los objetos creados tendrán efectividad y durabilidad máximas.

Con esto terminamos con los niveles de habilidad.

CONCLUSIONES

En este apartado se exponen las conclusiones del proyecto desde el punto de vista de análisis del trabajo realizado.

Durante el desarrollo de este proyecto se han cumplido todos los requisitos estipulados por el cliente. Teniendo en cuenta que este proyecto no es un encargo fijo,

sino que se corresponde más con una tarea de diseño de soluciones, los requisitos eran muy laxos y, mediante numerosas reuniones con el cliente y el diseñador de la parte física del juego, se ha conseguido idear un sistema que satisface las necesidades del cliente.

Los aspectos más desafiantes han sido sin duda los diseños del inventario, que se han revisado numerosas veces según surgían nuevas necesidades que dejaban la anterior versión obsoleta.

Los algoritmos de combinación de objetos también han resultado una parte interesante a diseñar e implementar, dado que implicaban muchos factores.

Con este proyecto se ha aprendido a trabajar a contrarreloj en un proyecto en el que los requisitos, aunque muy laxos, cambiaban sin cesar. El equipo ha tenido que adaptarse a esta variabilidad implementando la aplicación de manera reutilizable y extremadamente modular, de manera a que estos cambios no afectaran demasiados aspectos y no obligaran a una reingeniería.

También ha servido para tener una experiencia de trato directo con el cliente y una profunda experiencia de análisis y resolución de problemas.

TRABAJOS FUTUROS

En esta sección se explican los trabajos que se desean realizar posteriormente al desarrollo de este documento.

INTEGRACIÓN DEL MÓDULO DE HABILIDADES

El módulo de habilidades está diseñado para afectar a numerosos campos de la lógica de esta aplicación. Un trabajo a realizar es la implementación de los métodos pertinentes para la integración de este módulo.

Por ejemplo las habilidades deberán afectar al módulo de creación de objetos, al módulo de conflictos, al módulo de informática, etc.

INTEGRACIÓN CON EL SERVIDOR

Se desea incorporar un servidor a la aplicación para la gestión de datos de usuario, inicio de sesión, funcionalidad en línea.

Aunque parte del servidor ya ha sido implementada, se precisa una extensa tarea de integración del mismo con la aplicación.

IMPLEMENTACIÓN DEL SISTEMA DE COMBATES

El sistema de combates, aunque diseñado (en versión alfa), ha de implementarse incluyendo esto las tareas de integración con los módulos actuales. Es probable que la implementación de este módulo implique también una extensión o remodelado de la base de datos local.

IMPLEMENTACIÓN DE LA FUNCIONALIDAD EN LÍNEA

Esta probablemente sea la tarea más fundamental en cuanto a criterio de éxito del proyecto. Es preciso implementar la funcionalidad en línea de la aplicación permitiendo conflictos, ataques informáticos y mensajería entre jugadores.

AÑADIDO DE FUNCIONALIDAD EN LOS DISTINTOS MÓDULOS YA IMPLEMENTADOS

Conforme surjan nuevas necesidades del cliente o del diseñador del juego de mesa, es probable que deban implementarse nuevas funcionalidades en los módulos ya existentes.

INTEGRACIÓN DE LOS DISEÑOS GRÁFICOS

Se deberán integrar los diseños gráficos encargados a un diseñador en la aplicación.

REFACTORING

Realizar tareas de refactoring para eliminar ciertos defectos del código como los mencionados del atributo quantity.

INTERNACIONALIZACIÓN

Deberán llevarse a cabo tareas de internacionalización de la aplicación.

MANTENIMIENTO

Se deberá dar un mantenimiento a la aplicación.

GLOSARIO

- **Android**: Sistema operativo para smartphones desarrollado por Google.
- **API**: "Application Programming Interface" por sus siglas en inglés; es un conjunto de funciones o métodos que ofrecen una determinada funcionalidad para el desarrollo de software. Puede ser considerado como una librería.
- **MVC**: Patrón de diseño Modelo – Vista – Controlador. Es un patrón de diseño de software.
- **Modelo**: En el patrón modelo – vista – controlador (MVC), el modelo se refiere a los datos que usa un software. A menudo el concepto de persistencia es usado para referirse a estos datos.
- **Vista**: En el patrón modelo – vista – controlador (MVC), la vista se refiere a la interfaz con la que interactúa el usuario final.
- **Controlador**: En el patrón modelo – vista – controlador (MVC), el controlador se refiere a los métodos encargados de gestionar las interacciones del usuario sobre la vista.
- **Persistencia**: La persistencia de una aplicación se refiere a los datos, normalmente guardados en una base de datos, que son persistentes a lo largo del tiempo de vida de un software.
- **Base de datos**: En el entorno de la programación de software, una base de datos es el lugar en el que se almacenan los datos persistentes de la misma.
- **Objeto:** En Java, un objeto es la instanciación de una clase Java.
- **Clase:** En java, una clase es el esqueleto mediante el cual se instancian los objetos. En una clase se definen los atributos y métodos que compondrán los objetos.
- **Layout:** En Android y en numerosos entornos de programación de interfaces visuales, un layout es el código que define la estructuración de esta interfaz.
- **Activity o Actividad:** En Android una actividad se corresponde (a grandes rasgos) con las pantallas de la aplicación.
- **Intent:** En Android, un intent es el mecanismo programático usado para iniciar nuevas actividades.
- **Kickstarter:** Plataforma de crowdfunding
- **Tiempo muerto:** En el ámbito del juego de mesa que está relacionado con este proyecto, un tiempo muerto es aquel en el que el jugador no está en su turno activo.
- **Tiempo activo:** En el ámbito del juego de mesa que está relacionado con este proyecto, un tiempo activo es aquel en el que el jugador está en su turno activo.

- **Código QR:** Un código QR, Quick-Response code por sus siglas en inglés, es un código descifrable por la cámara de un teléfono que contiene cierta información.
- **Ciclo de vida:** En informática, son las fases por las que pasa un software desde su concepción hasta el fin de su uso.
- **GNU:** De sus siglas en inglés GNU's Not Unix, es un sistema operativo
- **GPL:** General Public License, licencia de software.
- **LGPL:** Lesser General Public License, licencia de software.
- **MIT License:** Massachusetts Institute of Technology, licencia de software.
- **BSD License:** Berkley Software Distribution, licencia de software.
- **Paquete:** En programación java, es una agrupación de clases.
- **PVP:** Player Vs Player.
- **PVE:** Player Vs Environment.
- **NPC:** Non Playable Character. Este término usado en la industria de los videojuegos hace referencia a los personajes que no controla el jugador. En nuestro caso los personajes no jugables o NPC hacen referencia a los enemigos que aparecen bajo forma de carta.

BIBLIOGRAFÍA

[1] LunarSoundDesign, «2014 - U.S. Board Game Release Schedule,» 3 Diciembre 2013. [En línea]. Available: http://boardgamegeek.com/geeklist/166221/2014-us-board-game-release-schedule. [Último acceso: 1 Julio 2014].

[2] «IDC: Smartphone OS Market Share 2014, 2013, 2012, and 2011,» [En línea]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp#. [Último acceso: 1 Julio 2014].

[3] Wikipedia, «Java bytecode instruction listings,» [En línea]. Available: http://en.wikipedia.org/wiki/Java_bytecode_instruction_listings. [Último acceso: 1 July 2014].

[4] H. J, «Samsung sells more smartphones than all major manufacturers combined in Q1 | SamMobile,» [En línea]. Available: http://www.sammobile.com/2014/05/01/samsung-sells-more-smartphones-than-all-major-manufacturers-combined-in-q1/.

[5] R. Amadeo, «Google's iron grip on Android: Controlling open source by any means necessary | Ars Technica,» [En línea]. Available: http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/.

[6] Android developers, «Dashboards,» [En línea]. Available: https://developer.android.com/about/dashboards/index.html.

[7] Android developers, «Activity,» [En línea]. Available: http://developer.android.com/reference/android/app/Activity.html.

[8] D. Olsson, «Android: Clear Activity Stack,» [En línea]. Available: http://stackoverflow.com/questions/7075349/android-clear-activity-stack.

[9] Wikipedia, «Micromecenazgo,» [En línea]. Available: http://es.wikipedia.org/wiki/Micromecenazgo.

[10] H. S. LLC, «Golem Arcana,» Harebrained Schemes LLC, 13 Octubre 2013.. [En línea]. Available: https://www.kickstarter.com/projects/1613260297/golem-arcana.

[11 D. Waves, «QR Code Features,» [En línea]. Available:

] http://archive.today/20120915040047/http:/www.qrcode.com/en/qrfeature.html.

[12] Wikipedia, «QR Code,» [En línea]. Available: http://en.wikipedia.org/wiki/QR_code.

[13] A. Tarantola, «How QR Codes Work,» [En línea]. Available: http://gizmodo.com/5969312/how-qr-codes-work-and-why-they-suck-so-hard.

[14] S. B. Wicker y V. K. Bhargava, Reed-Solomon codes and their applications, John Wiley & Sons, 1999.

[15] Oracle, «HashMap (Java Platform SE 6),» [En línea]. Available: http://docs.oracle.com/javase/6/docs/api/java/util/HashMap.html.

[16] A. T. P. Site, «Non-constant Fields in Case Labels,» [En línea]. Available: http://tools.android.com/tips/non-constant-fields.

[17] Wikipedia, «Class-based programming,» [En línea]. Available: http://en.wikipedia.org/wiki/Class-based.

[18] Wikipedia, «Concurrent computing,» [En línea]. Available: http://en.wikipedia.org/wiki/Concurrent_computing.

[19] Wikipedia, «Object-oriented programming,» [En línea]. Available: http://en.wikipedia.org/wiki/Object-oriented_programming.

[20] Wikipedia, «Java (programming language),» [En línea]. Available: http://en.wikipedia.org/wiki/Java_(programming_language).

[21] Wikipedia, «Java bytecode,» [En línea]. Available: http://en.wikipedia.org/wiki/Java_bytecode.

[22] Biocram, «Integrate ZXing as a library inside an Android project,» [En línea]. Available: http://biocram.wordpress.com/2013/06/11/integrate-zxing-as-a-library-inside-an-android-project/.

[23] D. Flannery, «Integrate zxing barcode scanner into your Android app natively using Eclipse,» [En línea]. Available: http://damianflannery.wordpress.com/2011/06/13/integrate-zxing-barcode-scanner-into-your-android-app-natively-using-eclipse/.

[24] Wikipedia, «Modelo–vista–controlador,» [En línea]. Available:

] http://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador.

[25 J. Gosling, B. Toy, G. Steele, G. Bracha y A. Buckley, «The Java® Language
] Specification Java SE 8 Edition,» 03 Marzo 2014. [En línea]. Available:
http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf.

[26 AndroidAZ, «ZXing QR Reader Direct Integration,» [En línea]. Available:
] http://www.androidaz.com/development/zxing-qr-reader-direct-integration.

[27 Wikipedia, «Android (operating system),» [En línea]. Available:
] http://en.wikipedia.org/wiki/Android_(operating_system).

[28 The Apache Software Foundation, «Apache License and Distribution FAQ,» [En
] línea]. Available: http://www.apache.org/foundation/license-faq.html.

[29 The Apache Software Foundation, «Apache License, Version 2.0,» [En línea].
] Available: http://www.apache.org/licenses/LICENSE-2.0.html.

[30 Android developers, «App Manifest,» [En línea]. Available:
] http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[31 Android developers, «Intent,» [En línea]. Available:
] http://developer.android.com/reference/android/content/Intent.html.

[32 Wikipedia, «BSD licenses,» [En línea]. Available:
] http://en.wikipedia.org/wiki/BSD_licenses.

[33 Free Software Foundation, «GNU Lesser General Public License,» [En línea].
] Available: https://www.gnu.org/licenses/lgpl.html.

[34 Wikipedia, «GNU Lesser General Public License,» [En línea]. Available:
] http://es.wikipedia.org/wiki/GNU_Lesser_General_Public_License.

[35 Free Software Foundation, «GNU General Public License,» [En línea]. Available:
] http://www.gnu.org/licenses/gpl-3.0.en.html.

[36 Wikipedia, «GNU General Public License,» [En línea]. Available:
] http://en.wikipedia.org/wiki/GNU_General_Public_License.

[37 Free Software Foundation, «Frequently Asked Questions about the GNU
] Licenses,» [En línea]. Available: http://www.gnu.org/licenses/gpl-faq.en.html.

[38 Open Source Initiative, «The MIT License (MIT) | Open Source Initiative,» [En

] línea]. Available: http://opensource.org/licenses/MIT.

[39 Wikipedia, «MIT License,» [En línea]. Available:
] http://en.wikipedia.org/wiki/MIT_License.

[40 M. L. Murphy, The Busy Coder's Guide to Android Development, United States:
] CommonsWare, 2008.

ANEXO A: DISEÑO DEL SISTEMA DE CONFLICTOS

En este anexo se describe el sistema de combate del juego que, por no pertenecer a la primera fase del proyecto, no se ha incluido en el documento.

Note que esto sólo es el diseño del sistema, la implementación no se contempla en esta fase. El diseño está en fase alfa y, por tanto, está sujeto a modificaciones.

El sistema de combate está diseñado para funcionar en red entre dos dispositivos que ejecuten la aplicación o contra un enemigo del propio juego de mesa, es decir, un NPC. En ningún caso se realizarán conflictos que impliquen a más de dos actores.

PVP (PLAYER VS PLAYER)

El sistema de combate entre jugadores tendrá en cuenta los siguientes factores para resolver los conflictos:

- La habilidad del jugador en conflictos
- La carta de acción utilizada por el jugador
- El arma equipada por el jugador

El oponente no tiene ninguna información sobre estos factores (a menos que haya utilizado su dispositivo para robar cierta información, pero eso es otro tema) con lo cual se consigue un sistema de combate en el que ambos jugadores deberán "arriesgarse" a gastar una de sus cartas de acción de alto valor con el fin de asegurarse la victoria.

PVE (PLAYER VS ENVIRONMENT)

El sistema de combates contra un enemigo del juego de mesa tiene en cuenta los mismos factores para calcular la fuerza del jugador. En el caso del enemigo no jugador, éste tendrá un rango que será visible por el jugador. De esta forma el jugador sabrá que puede ser atacado dentro de unos ciertos valores y tendrá que ponderar si vale la pena o no gastar una carta de acción elevada.

RESOLUCIÓN

La resolución de conflictos se realiza en dos fases: fase de diálogo y fase de conflicto.

Automáticamente al entrar en conflicto, este se intenta resolver mediante un diálogo. Los diálogos funcionan exactamente igual que los conflictos excepto que no se tiene en cuenta el objeto equipado.

En caso de fracasar el diálogo se pasa a la fase de conflicto.

En ambos casos la resolución es muy sencilla, el combatiente que saque mayor fuerza de ataque ganará el conflicto.

En caso de ser fase de conflicto, el daño infligido se calcula como la diferencia entre las fuerzas de ataque de los oponentes.

Por ejemplo si un jugador A ataca a B por un valor de 7 y el jugador B por un valor de 5, el jugador B recibirá 2 puntos de daño.

ANEXO B: LICENCIAS DE SOFTWARE LIBRE

Nuestra aplicación hace uso de la librería ZXing de Google. Por ello se considera relevante una pequeña exposición sobre algunas licencias de software libre, qué implican y cuales nos convienen para un proyecto comercial.

LICENCIA APACHE VERSIÓN 2[9]

La librería ZXing para el escaneo de códigos QR esta licenciada bajo Apache versión 2.

Esta licencia es una licencia de software libre que permite el uso y reproducción del software bajo ciertas condiciones.

Permite:

- Descargar y usar libremente la parte o la totalidad del software para usos personales, de compañía o comerciales.
- Usar el software en creaciones propias, ya sea en paquetes o distribuciones.

Prohíbe:

- Redistribuir el software bajo licencia sin atribuir la autoría del software a su creador.
- Usar cualquier marca propiedad de apache dando a entender que la fundación apache respalda el nuevo software distribuido.
- Atribuirse la autoría del software bajo licencia Apache.

Requiere:

- Incluir una copia de la licencia original del software licenciado en cualquier distribución que use dicho software.
- Indicar de manera clara que se está haciendo uso de software Apache, señalando la autoría del mismo.

No requiere:

- Incluir una copia del código fuente del software Apache original o modificado en la distribución.
- Señalar los cambios hechos en el software de Apache a la fundación Apache.

Esta licencia es muy apropiada para nuestro proyecto ya que permite usar el software de Apache en un producto comercial. En nuestro caso hemos de prestar especial

[9] [29] [28]

atención en incluir una copia de la licencia en el apartado “acerca de” de nuestra aplicación.

LICENCIA GNU GPL[10]

La licencia GNU GPL (General Public License) es una licencia de software libre que permite el uso, la modificación y la redistribución de software.

GPL permite que el autor del código bajo software GPL cobre una tasa por el software. La FSF (Free Software Foundation) dice que el software libre no debería poner restricciones al uso comercial y la liciencia GPL dice explícitamente que los trabajos bajo GPL pueden ser vendidos a cualquier precio.

GPL también estipula que aquellos que usen o redistribuyan software GPL no deben imponer más restricciones que las propias de la licencia GPL. Esto prohíbe distribuir el software bajo términos de uso como un acuerdo de no-divulgación o contratos.

Con la licencia GPL, se obliga a que versiones pre-compiladas del software que use o sea modificación de software GPL venga acompañado de una copia del código fuente, una disposición escrita a redistribuir el código fuente de la misma forma que el código fuente, o una oferta por escrito para obtener el código fuente del código bajo GPL. También se solicita que venga una copia de la licencia GPL.

La FSF no tiene derechos de copyright sobre el software GPL a menos que el autor explícitamente asigne estos derechos a la FSF. Sólo los usuarios que tengan el derecho de copyright tienen derecho a denunciar cuando una violación de los términos de la licencia ocurre.

Para usos exclusivamente privados, podrá usar y modificarse el código fuente bajo licencia GPL sin la necesidad de publicar el código fuente del mismo. En usos comerciales, es obligatorio publicar el código fuente en su totalidad, incluyendo cualquier modificación aportada al mismo.

En caso de que un programa use código GPL y sea distribuido, deberá, en su totalidad, estar conforme a la licencia GPL.

LICENCIA GNU LGPL[11]

La licencia LGPL (Lesser General Public Licence) es una derivación de la licencia GPL.

[10] [35] [37] [36]
[11] [33] [34] [37]

La principal diferencia es que los programas que usen partes de códigos bajo licencia LGPL no tienen que acogerse a la licencia LGPL, pudiendo ser programas privados.

Sólo en caso de ser una derivación del software original deberá acogerse a la licencia LGPL. Un ejemplo típico de un programa no considerado derivativo es aquel que hace uso de una librería LGPL.

LICENCIA MIT[12]

La licencia MIT es una licencia de software libre originaria del Masachusetts Institute of Technology. Es una licencia permisiva que permite el uso de software dentro de software privado siempre y cuando las licencias del software que hace uso de software MIT contengan una copia de los términos de uso de la licencia MIT.

El software resultante no pierde su naturaleza privada aunque incorpore software bajo licencia MIT. No se solicita que se incorpore una copia del código fuente del programa, pudiendo distribuirse binarios sin ningún impedimento.

LICENCIA BSD[13]

Las licencias BSD (Berkley Software Distribution) son licencias de software libre permisivas, muy parecida a la licencia MIT, que permiten el uso de su software en programas propietarios y no solicitan la divulgación del código fuente.

Existen 4 tipos de licencias BSD:

- **Licencia previa:** Esta licencia, previa a la licencia de 4 términos, sólo solicita que se incorpore una copia del párrafo de la licencia en todas las partes necesarias de manera a que se asegure que se reconoce la autoría del software.
  También se prohíbe usar el nombre de la compañía desarrolladora del software bajo licencia de manera que parezca que esta respalda el nuevo software.
- **Licencia de 4 cláusulas**: Esta licencia incorpora las siguientes 4 clausulas:
  - La redistribución del código fuente debe mantener una copia del copyright.
  - La redistribución en formato binario debe incorporar una copia de la licencia en la documentación o cualquier otro material entregado con la distribución.
  - Cualquier material publicitario que mencione funcionalidades o usos de este software debe incluir la siguiente frase:
    “Este producto incluye software desarrollado por <organización>

[12] [38] [39]
[13] [32]

    - Ni el nombre de la organización ni el de sus contribuidores podrá usarse para respaldar productos derivados de este software.
- **Licencia de 3 clausulas:** Esta licencia elimina la tercera cláusula de la licencia de 4 cláusulas dado que esta presentaba problemas de compatibilidad con la licencia GPL.
- **Licencia de 2 cláusulas**: La licencia de 2 cláusulas elimina la 3ª y 4ª cláusula de la licencia de 4 cláusulas y añade más texto al final de la licencia sobre opiniones expresadas sobre el software.

ANEXO C: INTEGRACIÓN DE UN LECTOR QR EN UNA APLICACIÓN ANDROID[14]

Existen numerosos lectores QR en el mercado. Todos ellos realizan la misma función: descodificar códigos QR y sacar la información de los mismos. Para nuestra aplicación era preciso incorporar un lector de códigos, a ser posible, offline.

Después de un trabajo de investigación en internet, se estimó que la librería ZXing de Google era la más adecuada para nuestro proyecto.

A continuación vamos a ver brevemente cómo añadir un proyecto externo como librería a nuestro proyecto Android.

1. En primer lugar hemos de obtener el código fuente. En nuestro caso está ubicado en https://github.com/zxing/zxing
2. Una vez descargado, habremos de construir el último "snapshot". Para ello utilizaremos MAVEN y ejecutaremos mvn package en la carpeta core del proyecto.
3. Ahora tendremos que crear un proyecto de eclipse a partir del código descargado.

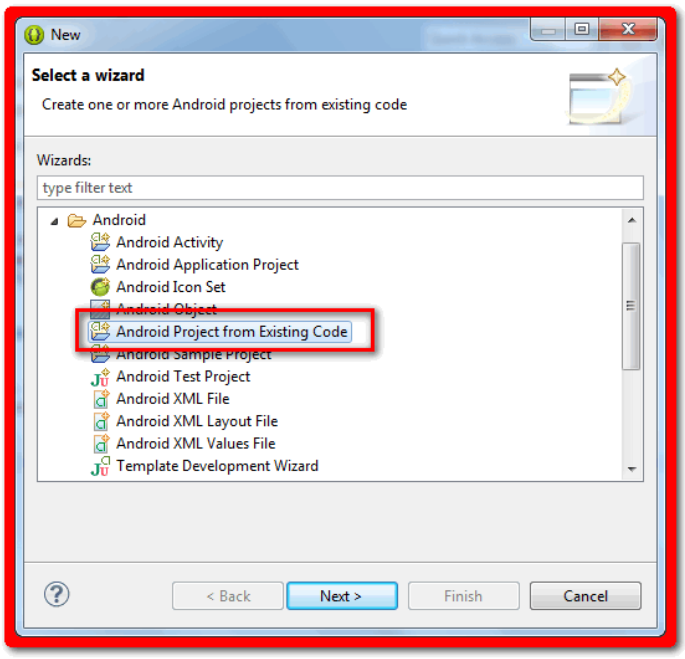

ILUSTRACIÓN 24: CREAR PROYECTO ANDROID A PARTIR DE CÓDIGO EXISTENTE

4. Una vez tengamos el proyecto creado, tenemos que incluir el archivo .jar creado en el paso 2.

[14] [26] [22] [23]

5. Una vez hecho esto, tendremos que ir a las propiedades del proyecto y marcar este proyecto como librería:

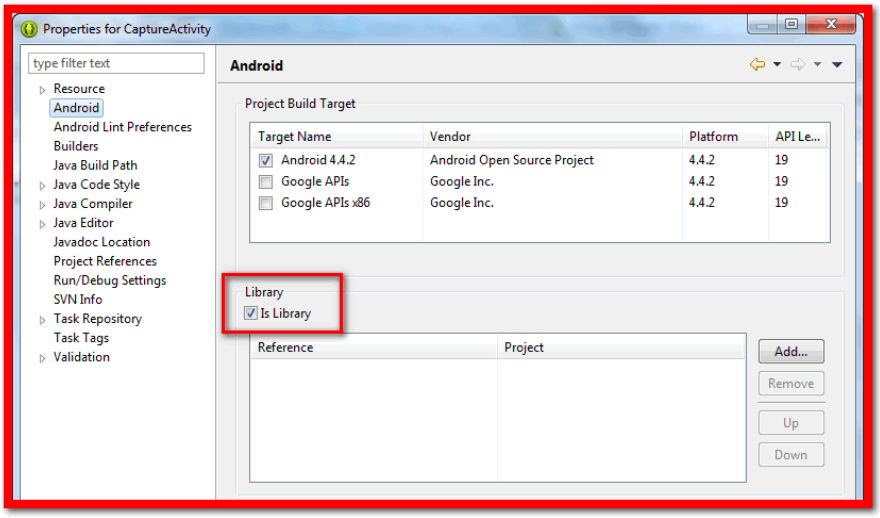

ILUSTRACIÓN 25: MARCAR PROYECTO COMO LIBRERÍA

6. Ahora tendremos que convertir todas las sentencias “switch” que se ejecutan sobre variables a sentencias if-else. Por suerte eclipse se encarga de hacer esto, sólo tendremos que ir a cada error que nos salga y pedir a eclipse que nos lo convierta [16].
7. Una vez hecho esto, sólo tenemos que añadir una referencia a la librería en nuestro proyecto a través de las propiedades.

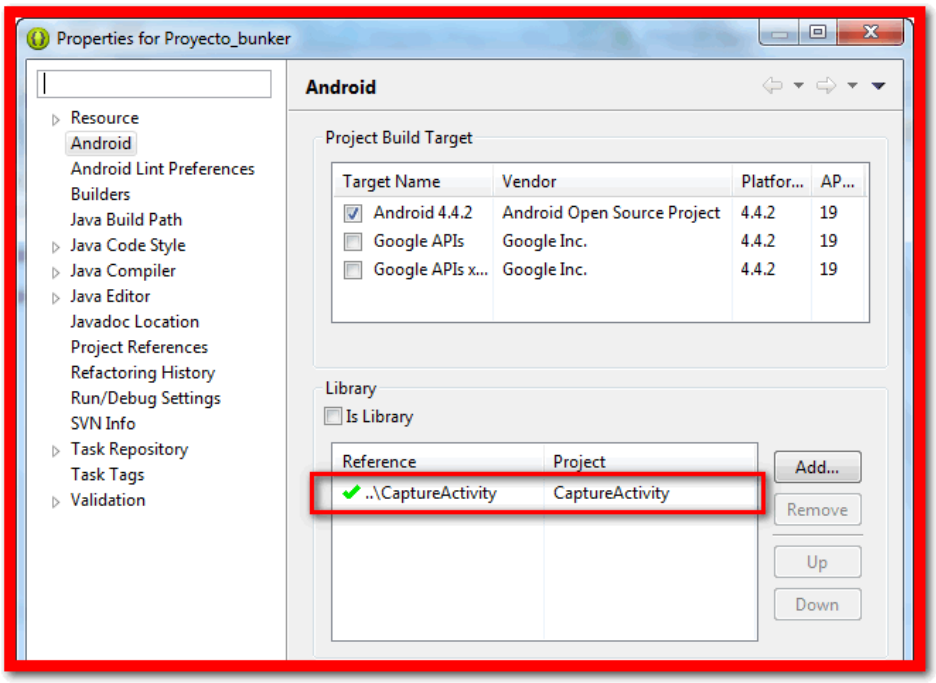

ILUSTRACIÓN 26: AÑADIR REFERENCIA A UNA LIBRERÍA EXTERNA

Con esto ya habríamos integrado el lector de códigos a nuestra aplicación.

ANEXO D: ARGUMENTO Y FUNCIONAMIENTO DEL JUEGO DE MESA

ARGUMENTO

Esta es una obra de ficción, el autor de este documento no acepta ni rechaza ninguno de los conceptos descritos en la siguiente historia.

En el año 1990, a causa de la contaminación, el agua es portadora de un virus que contamina progresivamente a la población del planeta.

Las distintas potencias del mundo inician un proyecto de investigación del virus, construyendo para ese fin distintos laboratorios dispersados por el planeta. Estos laboratorios están protegidos por las fuerzas armadas y apartados de la civilización.

Los gobiernos toman la decisión de ocultar la existencia del virus a la población ya que este no parece manifestarse hasta unos diez o quince años después de la infección.

Diez años más tarde se produce lo que se conoce como el "efecto 2000", todos los dispositivos electrónicos se desconfiguran, generando caos. Se producen varios ataques informáticos a las distintas organizaciones de inteligencia mundiales que revelan la existencia del virus. Empieza entonces una etapa de revueltas y enfrentamientos de los gobiernos con grupos antisistema. Además, los efectos del virus empiezan a hacerse notar en la población, generando comportamientos agresivos, pérdida de identidad, psicosis, etc.

Mientras tanto se descubre una forma de ralentizar los efectos del virus gracias a un suero, sin embargo éste solo se suministrará a aquellos ciudadanos que acepten someterse a una cirugía experimental en la que se implantará un dispositivo de seguimiento para monitorizar al sujeto. Este dispositivo, llamado IDlog, se implanta en el brazo del sujeto y va conectado al cerebro.

Con el fin de obtener sujetos de prueba, aquellos ciudadanos que se sometan al programa de seguimiento del gobierno vivirán con una serie de "comodidades" como comunicación con familiares, controles de salud, etc.

A aquellas personas que se sometieron a la cirugía se les proporcionó un lugar seguro en distintos bunkers construidos por todo el planeta en los cuales se distribuye agua potable libre de infecciones y donde se asegura la protección de los ciudadanos ante amenazas exteriores. Estas amenazas toman la forma de contacto directo con el virus, saqueadores, grupos de la resistencia, animales y humanos contagiados, etc.)

En el año 2014, un centro de investigación emite un mensaje en el que asegura haber encontrado la cura definitiva para el virus. Sin embargo todos los consiguientes intentos de ponerse en contacto con el centro fracasan, dejando paso al silencio.

Cinco de los bunkers más cercanos envían a una persona para intentar ponerse en contacto con el centro de investigación.

En este punto empieza el juego.

FUNCIONAMIENTO

En esta sección vamos a hacer un repaso extremadamente breve sobre el funcionamiento del juego de mesa con el fin de que el lector entienda mejor el papel de la aplicación.

Ni que decir tiene que cada jugador encarna uno de los cinco enviados para comunicarse con el centro de investigación.

El juego de mesa está pensado para ser de 3 a 5 jugadores.

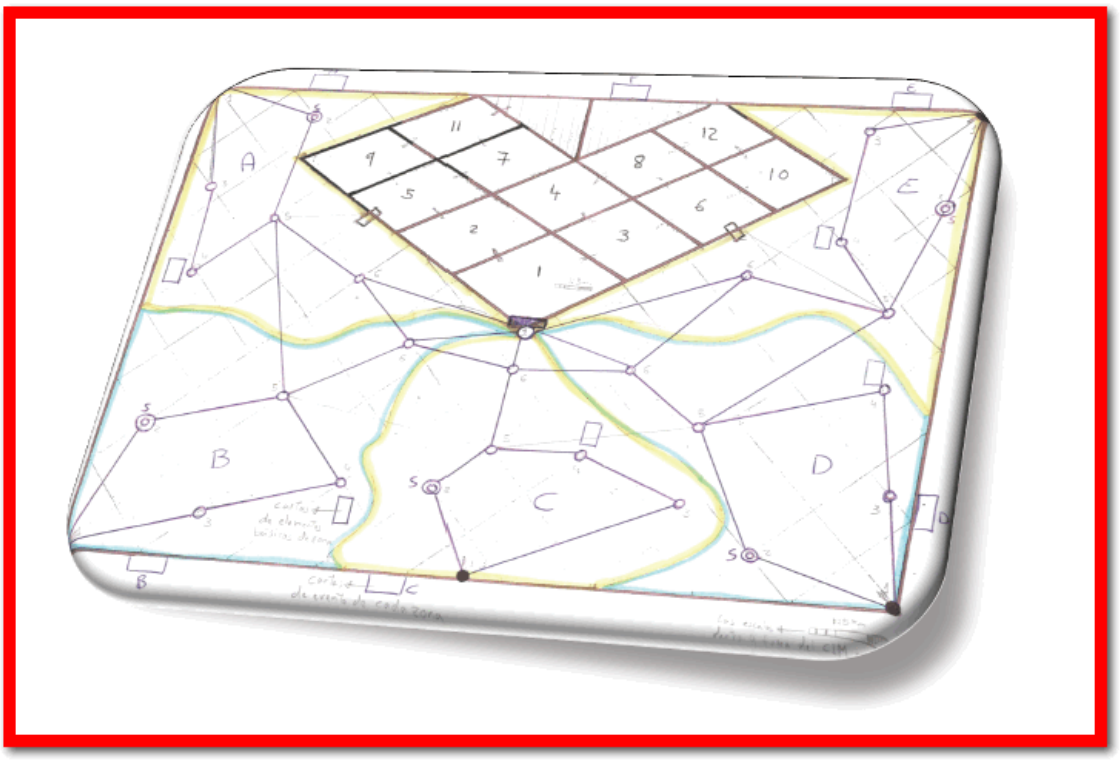

ILUSTRACIÓN 27: TABLERO DEL JUEGO DE MESA

En la [Ilustración 27] puede verse un esquema del tablero.

El tablero representa el mundo conocido por los jugadores. En las esquinas y en la parte central inferior se ubican los bunkers de inicio. En la parte central superior aparece el centro de investigación militar (CIM) objetivo de los jugadores.

El CIM está a una escala distinta que el resto del mapa.

El tablero se divide en nodos y los jugadores se desplazarán de nodo a nodo. Desplazarse de un nodo a otro tiene un coste asociado. Así se ha conseguido diseñar un tablero en el que independiente del lugar de partida de un jugador, éste podrá dirigirse al CIM con el mismo coste que sus competidores.

El CIM representa una parte diferente del resto del tablero y tiene un funcionamiento diferente: Si bien el concepto del movimiento es el mismo, la parte del CIM se desplegará con cuadrados boca abajo en cada partida. De esta forma se consigue una cierta aleatoriedad en el interior del CIM, dando algo particular a cada partida.

El tablero está dividido en cinco zonas, excluyendo al CIM, cada una con características propias.

El objetivo de cada jugador es llegar al CIM para encontrar la cura al virus. Al iniciarse la partida cada jugador deberá dirigirse hacia allí sorteando los peligros del camino.

A lo largo del camino los jugadores irán encontrando objetos que les ayudarán en su causa, al igual que pueden encontrarse con conflictos que les ralentizarán.

Los jugadores son, en principio, hostiles entre ellos, permitiéndose el conflicto. Los jugadores también podrán aliarse para conseguir objetivos comunes, siempre que lleguen a un acuerdo.

La aplicación representa el dispositivo que los ciudadanos de los bunkers tienen implantado en el brazo. Mediante la aplicación se pretende que los jugadores puedan gestionar su inventario, combinar objetos, resolver conflictos, comunicar con otros jugadores, piratear otros dispositivos y gestionar sus habilidades u “aumentos”.

Los apartados de inventario, combinación de objetos, resolución de conflictos y habilidades ya han sido explicados previamente, la comunicación entre dispositivos no está diseñada aún así que no nos atrasaremos con ella. El pirateo de dispositivos merece una pequeña mención.

El pirateo de dispositivos servirá para inutilizar dispositivos ajenos, obtener información de los mismo como, por ejemplo el estado de salud mental y física de un jugador, el arma equipada, los mensajes que han sido intercambiados con otros jugadores, etc. Para ello será necesario que el jugador que desee piratear tenga en su

posesión un programa para ello. Los programas están repartidos por el mundo bajo forma de diskettes y funcionan de forma muy parecida a los objetos.

Existen programas ofensivos y defensivos, los programas defensivos, o “firewalls”, sirven para defenderse de los ataques informáticos. Una vez cargados en el sistema, reducen la vulnerabilidad del mismo bloqueando una serie de ataques.

Un programa ofensivo tiene asociado un valor de potencia o fuerza que le permite superar determinados firewalls. Siempre existe una probabilidad de que el ataque falle, así como también existe la posibilidad de que un ataque de grado 3 que se enfrenta a un firewall de grado 4 funcione.

Hasta aquí nuestra breve exposición sobre el funcionamiento del juego de mesa y de algunos mecanismo adicionales.

ANEXO E: CARGA DE TRABAJO

Este proyecto está enfocado al desarrollo de la lógica de una aplicación y de su integración con el mundo físico. El resto de la aplicación, es decir la interfaz y el diseño de las bases de datos es menester de otro proyecto.

En este anexo se detallará la carga de trabajo de este proyecto en conjunción con el siguiente proyecto:

"*Diseño de una interfaz para una aplicación así como el control táctil de la misma y diseño de las estructuras de datos necesarias para el funcionamiento de la aplicación*"

Código del proyecto: ISSI_119/1314

Tarea	ISSI_119/1314	SIIS_120/1314
Captura y análisis de requisitos	X	X
Diseño de las bases de datos	X	
Diseño de la interfaz de usuario	X	
Diseño e implementación del servidor	X	
Integración de un lector QR		X
Integración con el entorno físico		X
Diseño de clases		X
Diseño e implementación de la lógica del juego		X
Implementación de los métodos de acceso a la información	X	
Gestión de recursos	X	X
Pruebas	X	X

TABLA 2: CARGA DE TRABAJO